



# **Assessing the Impact of Development Disruptions and Dependencies in Analysis of Alternatives of System-of-Systems**

**Final Technical Report SERC-2012-TR-035**

December 31, 2012

Principal Investigator: Dr. Daniel A. DeLaurentis, Purdue University

Co-Principal Investigator: Dr. Karen Marais, Purdue University

## **Team Members**

Navindran Davendralingam, Postdoctoral Researcher, Purdue University

Seung Yeob Han, PhD Student, Purdue University

Payuna Uday, PhD Student, Purdue University

Zhemei Fang, PhD Student, Purdue University

Cesare Guariniello, PhD Student, Purdue University

Report Documentation Page		Form Approved OMB No. 0704-0188
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.		
1. REPORT DATE <b>31 DEC 2012</b>	2. REPORT TYPE	3. DATES COVERED <b>00-00-2012 to 00-00-2012</b>
4. TITLE AND SUBTITLE <b>Assessing the Impact of Development Disruptions and Dependencies in Analysis of Alternatives of System-of-Systems</b>		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER	
	5e. TASK NUMBER	
	5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) <b>Purdue University, West Lafayette, IN, 47907</b>		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT <b>Approved for public release; distribution unlimited</b>		
13. SUPPLEMENTARY NOTES		

#### 14. ABSTRACT

The development of a System-of-Systems (SoS) remains a highly challenging endeavor due to the complex interdependencies between systems that often times exhibit managerial and operational independence, yet, must work cohesively to achieve an overarching set of capabilities. Current guidelines set forth by the Department of Defense SoS System Engineering guide presents SoS SE as a set of seven core elements which are connected to the 16 technical and management processes in the Defense Acquisition Guidebook (DAG). This guide, however, and subsequent frameworks (such as the Wave Model), are meant to raise awareness of the key issues and products involved. A need exists to create and mature decision support tools to support the decision making process of evolving SoS architectures; this includes the need for properly assessing the impact that potential disruptions can have, and the analysis of alternatives of SoS constructs. Trades between capability and risk are essential decisions that must be addressed for SoS capability planning. Existing tools for such trades, where they exist, can be ineffective and non-intuitive when size and/or interdependency complexity is high. These features create a tradeoff space between development risk and capability potential of a system. The objective of RT-36 was to explore analytical methods to quantify the impact of system interdependencies in the context of SoS capability development and use this to guide system engineering activity for SoS. A variety of approaches are investigated to provide a means to conduct analysis of alternatives while navigating the decision space that simultaneously considers the potential positive impacts of interdependencies (e.g., SoS capability) as well as the negative impacts (e.g. consequences of disruption in development) The research in RT-36 centered on seven analytical methods that have been adapted to support SoS architecting decisions and systems engineering of constituent systems. Since no single method or tool can fulfill all technical and managerial needs, the exploration of methods in this report use generic forms of problems faced by practitioners, focusing on inputs, outputs, and limitations in the context of support for the ?Wave? model for SoS architectural evolutions. The methods explored in this report show great promise in term of supporting practitioners in performing relevant analysis and evolutions of SoS architectures; these methods will be further matured and enhanced in follow-up research work under RT-44.

#### 15. SUBJECT TERMS

##### 16. SECURITY CLASSIFICATION OF:

a. REPORT  
**unclassified**

b. ABSTRACT  
**unclassified**

c. THIS PAGE  
**unclassified**

##### 17. LIMITATION OF ABSTRACT

**Same as  
Report (SAR)**

##### 18. NUMBER OF PAGES

**96**

##### 19a. NAME OF RESPONSIBLE PERSON

Copyright © 2012 Stevens Institute of Technology, Systems Engineering Research Center

This material is based upon work supported, in whole or in part, by the U.S. Department of Defense through the Systems Engineering Research Center (SERC) under Contract H98230-08-D-0171. SERC is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense.

#### NO WARRANTY

THIS STEVENS INSTITUTE OF TECHNOLOGY AND SYSTEMS ENGINEERING RESEARCH CENTER MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. STEVENS INSTITUTE OF TECHNOLOGY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. STEVENS INSTITUTE OF TECHNOLOGY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

This material has been approved for public release and unlimited distribution except as restricted below.

Internal use:\* Permission to reproduce this material and to prepare derivative works from this material for internal use is granted, provided the copyright and "No Warranty" statements are included with all reproductions and derivative works.

External use:\* This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other external and/or commercial use. Requests for permission should be directed to the Systems Engineering Research Center at [dschultz@stevens.edu](mailto:dschultz@stevens.edu)

\* These restrictions do not apply to U.S. government entities.

## ABSTRACT

---

The development of a System-of-Systems (SoS) remains a highly challenging endeavor due to the complex interdependencies between systems that often times exhibit managerial and operational independence, yet, must work cohesively to achieve an overarching set of capabilities. Current guidelines set forth by the Department of Defense SoS System Engineering guide presents SoS SE as a set of seven core elements which are connected to the 16 technical and management processes in the Defense Acquisition Guidebook (DAG). This guide, however, and subsequent frameworks (such as the Wave Model), are meant to raise awareness of the key issues and products involved. A need exists to create and mature decision support tools to support the decision making process of evolving SoS architectures; this includes the need for properly assessing the impact that potential disruptions can have, and the analysis of alternatives of SoS constructs.

Trades between capability and risk are essential decisions that must be addressed for SoS capability planning. Existing tools for such trades, where they exist, can be ineffective and non-intuitive when size and/or interdependency complexity is high. These features create a tradeoff space between development risk and capability potential of a system. *The objective of RT-36 was to explore analytical methods to quantify the impact of system interdependencies in the context of SoS capability development and use this to guide system engineering activity for SoS.* A variety of approaches are investigated to provide a means to conduct analysis of alternatives while navigating the decision space that simultaneously considers the potential positive impacts of interdependencies (e.g., SoS capability) as well as the negative impacts (e.g. consequences of disruption in development)

The research in RT-36 centered on seven analytical methods that have been adapted to support SoS architecting decisions and systems engineering of constituent systems. Since no single method or tool can fulfill all technical and managerial needs, the exploration of methods in this report use generic forms of problems faced by practitioners, focusing on inputs, outputs, and limitations in the context of support for the “Wave” model for SoS architectural evolutions. The methods explored in this report show great promise in term of supporting practitioners in performing relevant analysis and evolutions of SoS architectures; these methods will be further matured and enhanced in follow-up research work under RT-44.

This Page Intentionally Left Blank

## TABLE OF CONTENTS

<b>Abstract .....</b>	<b>3</b>
<b>Table of Contents .....</b>	<b>5</b>
<b>Figures and Tables .....</b>	<b>7</b>
<b>1 Summary .....</b>	<b>10</b>
<b>2 Introduction (3-4 pages) .....</b>	<b>11</b>
<b>2.1 Introduction Subsection.....</b>	<b>11</b>
2.1.1 Introduction Sub-Sub Section .....	<b>Error! Bookmark not defined.</b>
<b>3 Analytic Methods .....</b>	<b>14</b>
<b>3.1 SoS Resilience Through Stand-In Redundancy.....</b>	<b>14</b>
3.1.1 Introduction.....	15
3.1.2 Analytic Framework .....	16
3.1.3 Application: An Illustrative Example.....	19
3.1.4 Conclusion and Future Work .....	24
<b>3.2 Evaluating SoS Resilience using Interdependency Analysis .....</b>	<b>25</b>
3.2.1 Introduction .....	25
3.2.2 Hierarchical Organization .....	26
3.2.3 An Interdependency Analysis of a System-of-Systems using a Bayesian Network.....	26
3.2.4 Conditional Resilience in the SoS Context .....	28
3.2.5 Total Resilience in the SoS Context .....	29
3.2.6 Application: A Case Study of Littoral Combat Ship System .....	30
3.2.7 Conclusion and Future Work.....	35
<b>3.3 Development Interdependency Modeling for System-of-Systems (SoS) : SoS Management Strategy .....</b>	<b>35</b>
3.3.1 Introduction .....	36
3.3.2 Overview of Development Interdependency Modeling for SoS .....	37
3.3.3 Inputs for Development Interdependency Modeling – Initial Development Failure Rates and Dependency Strength.....	37
3.3.4 An Interdependency Analysis of a System-of-Systems using a Bayesian Network.....	40
3.3.5 Update of Failure Rates for all activities.....	43
3.3.6 Application: A Synthetic Demonstration Problem .....	44
3.3.7 Conclusion and Future Work.....	47
<b>3.4 A Markov Model to Analyze Development Interdependencies in System-of-Systems .....</b>	<b>49</b>
3.4.1 Introduction .....	49
3.4.2 Analytic Framework.....	49
3.4.3 Application: An Illustrative Example .....	51
3.4.4 Conclusion and Future Work.....	52
<b>3.5 Dependency Network Analysis .....</b>	<b>52</b>
3.5.1 Introduction .....	<b>Error! Bookmark not defined.</b>
3.5.2 Functional Dependency Network Analysis.....	53
3.5.3 Development Dependency Network Analysis.....	60
3.5.4 Conclusions and Future Work .....	63
<b>3.6 Architecture Evolution Strategies Analysis using Colored Petri Nets .....</b>	<b>65</b>
3.6.1 Introduction .....	65

3.6.2 Background.....	66
3.6.3 Proposed Approach.....	67
3.6.4 Case Study.....	70
3.6.5 Conclusion and Future Work.....	73
<b>3.7 Robust Portfolio optimization Approach to Architecting System of Systems</b>	<b>74</b>
3.7.1 Introduction.....	74
<b>3.7.2 BACKGROUND AND MOTIVATION</b> .....	<b>75</b>
3.7.3 Robust Portfolio APPROACH: SoS Network Modeling.....	79
3.7.4 example Case Study: Littoral Combat Ship .....	83
3.7.5 Summary and Future Work .....	88
<b>4 Summary and Future Research.....</b>	<b>89</b>
<b>5 Bibliography .....</b>	<b>93</b>
<b>Appendices .....</b>	<b>Error! Bookmark not defined.</b>



## FIGURES AND TABLES

Figure 1: Wave model .....	11
Figure 1. Classification of SoS resilience based on when and how failures are addressed.....	16
Figure 2. (a) Physical (Han, Marais, & DeLaurentis, 2012) and (b) functional representation of system-of-systems.....	17
Figure 3. Notional LoP-LoR graph showing impact of stand-in redundancy on SoS performance .....	18
Figure 4. Five-system SoS and corresponding decomposition of capabilities .....	20
Figure 5. Systems and the functions they provide.....	20
Figure 6. Impact of system failure and stand-in redundancy on (a) LoP of Capability 1 (Surveillance), (b) LoP of Capability 2 (Target identification), and (c) LoP of Capability 3 (Target elimination).....	24
Figure 7. A hierarchical representation of a system-of-systems capability.....	27
Figure 8. A Bayesian Network representation.....	27
Figure 9. Performance-Failure diagram .....	29
Figure 10. Three different resilience patterns .....	30
Figure 11. Overall architecture of the LCS SoS (LCS: The USA's Littoral Combat Ships, 2012), (Czapiewski, 2004) .....	31
Figure 12. The decomposition of the LCS system.....	32
Figure 13. Conditional resilience values with one system failure in architecture 1 and 2.....	33
Figure 14. Resilience patterns with different failure rates of systems and different architectures .....	34
Figure 15. Evolution of the LCS SoS performances of both architectures for 600 minutes.....	35
Figure 16. Overview of the integrated simulation model for development interdependency analysis.....	37
Figure 17. Various shapes of beta distributions.....	38
Figure 18. Hierarchical representation of an entire system .....	39
Figure 19. A Bayesian Network representation .....	41
Figure 20. Uncertainty on the time steps .....	43
Figure 21. A very simple event tree structure for an example development schedule .....	44
Figure 22. A five system development network and all input information for the analysis .....	45
Figure 23. Information fusion of failure rates of system 2, 3, and 5 with system 1.....	45
Figure 24. Mean values for failure rates propagating to systems.....	46
Figure 25. (a) Mean values of integrated failure rate of system 4 with 95% confidence interval, (b) Mean values of integrated failure rate of system 4 in three different cases .....	47

Figure 26. A Markov network. F is the absorbing state, T's are the transition probabilities, D's are the weights (importance of the disruption being propagated through a link) .....	50
Figure 27. (a) Five-node whisper architecture, (b) Five-node blackboard architecture .....	51
Figure 28. Operational dependency of node $N_j$ from node $N_i$ .....	53
Figure 29. Correlation between Performance and Operability of a system. ....	53
Figure 30. A small network, showing the required input for FDNA .....	54
Figure 31. (a) the five-node aerospace SoS; (b) the same SoS represented as a network, with the required input for FDNA: self-effectiveness of each node, and COD and SOD of each link.	55
Figure 32. Probability distribution for the operability in the five-node system. (a) single failure in node $N_1$ , beta probability distribution for self-effectiveness of node $N_i$ ; the probability distribution of S2 coincides with that of S4; (b) failure in nodes $N_1$ and $N_4$ , with independent uniform distribution: node $N_3$ is in this case more resilient than node $N_5$ , differently from what appears in 32a .....	58
Figure 33. The twelve-nodes SoS.....	59
Figure 34. (a) the dependency between node $N_i$ and node $N_j$ . ....	61
Figure 35. Architecture modeling process.....	68
Figure 36. Functional representation of SUW architecture .....	70
Figure 37. Existing and evolving architecture alternatives .....	71
Figure 38. CPN model for existing architecture .....	71
Figure 39. Tradeoff space between performance and complexity.....	72
Figure 40: (a) SoS hierarchy (b) nodal behaviors.....	79
Figure 41: Littoral Combat Ship (LCS) .....	83
Figure 42 (a) Robust efficiency frontier (b) Multiple risk measures.....	86
Figure 44: (a) performance profile (b) portfolio of assets .....	88
Figure 45: SoS and Wave Model evolution.....	91
Table 1. Disruption and dependency effects to expected development time .....	46
Table 2. Node ranking by Markov analysis in whisper and blackboard architectures.....	52
Table 3. Degraded self-effectiveness in single systems. O=operability of the nodes .....	56
Table 4. Examples of degraded self-effectiveness in pairs of systems. O=operability. ....	57
Table 5. Twelve-node SoS experiment: indicated system is having a self-effectiveness of 20 (single failure in other systems gave the same results for all the experiments), while all other systems have self-effectiveness equal to 100 .....	59
Table 6. Preliminary results with DDNA method. BT=Beginning Time. CT=Completion Time. ....	63
Table 7. System performance and complexity calculation result .....	72
Table 8: LCS candidate systems .....	85

Table 9: Development covariance matrix .....	85
Table 10: LCS candidate systems.....	87

# 1 SUMMARY

---

Effective management of SoS architectures begins with careful consideration of the dynamics of SoS artifacts; however, the complex interdependencies of such design trade spaces across developmental and operational aspects of SoS makes effective management an elusive notion due to the lack of effective tools to navigate such complex trade spaces. Furthermore, the need for methods to balance risk and capability tradeoffs when assessing alternatives in decision making for SoS architectures has been a primary motivation for the research documented in this report this has resulted in the development of a suite of methods that address various issues faced in supporting SoS artifacts within the context of the Wave Model. The methods covered in this report include the following:

- **Robustness using Stand-In Redundancy**  
The method employed here employs robust strategies in evaluating and constructing SoS networks of assets that have the ability to mitigate
- **Interdependency and Resilience Analysis using Bayesian Networks**  
This method utilizes statistical/distribution information from data driven SoS networks to address issues of network capability resilience and analysis of interdependencies in developmental networks.
- **Interdependency analysis using Markov Networks**  
This method employs Markov network chains to analyze the impact of interdependencies on development of interconnected networks of systems.
- **Functional (Developmental) Dependency Network Analysis (F(D)DNA)**  
This method extends the Markov network approach to analysis of both functional and developmental dependencies between constituent systems in a SoS architecture.
- **Architectural development using Colored Petri Nets (CPN)**  
This method employs a discrete event simulation technique to analyze event based architectures; discrete event networks are modeled as a combination of tokens, transitions and rule sets.
- **SoS Architecture Decision Analysis using Robust Portfolio Optimization**  
This method provides a decision support framework for the identification of 'portfolios' of interdependent systems that fulfill requirements and capabilities.

The report is organized as follows: Section (2) of the report is the introduction and covers the background motivation behind the research in supporting SoS architectural evolution from the perspective of a SoS SE practitioner. Section (3) details the suite of researched analytic methods that are

adapted to support SoS architectural artifacts, as listed above. The description of each method includes background literature review, details of the methodology, adaptation to SoS problems and demonstration through application on a simplified case problem. These problems cover both generic nodal collection of systems and a Littoral Combat Ship (LCS) inspired case study.

## 2 INTRODUCTION

2.1 background/motivation The Department of Defense (DoD) has recognized the need for new principles in managing the development of SoS architectures. This has prompted the development and release of the SoS SE guidebook, which presents SoS SE as seven core elements that are mapped to the original 16 technical management processes within the Defense Acquisition Guidebook (DAG) (Defense, Defense Acquisition Guidebook, 2008). The guidebook serves as a concerted effort in understanding how SoS architectures work and address a holistic view of what frameworks are necessary in tackling SoS SE challenges. However, the guidance provided in the document is still in need of more comprehensive guidelines and complementary technical tools to enable effective SoS SE management and support.

More recent developments have seen extension and improvements in decision making , interrelationships and artifacts from a higher-level perspective, by altering the Trapeze Model perspective of viewing SoSs. A new paradigm of the ‘Wave Model’ as proposed by (Dahmann, Rebovich, & Lowry, May 2011) essentially unwinds the trapeze model construction in an effort to improve transparency and efficiency in SoS architectural issues. Figure 1 shows the “wave” model representation of SoS decision-making artifacts, as developed by (Dahmann, Rebovich, & Lowry, May 2011).

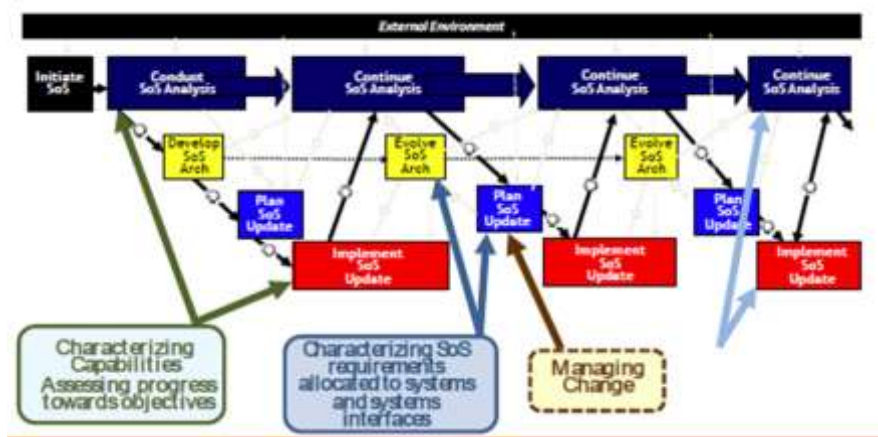


Figure 1: Wave model

Each step of decision epochs in the Wave Model, denoted by individually colored boxes in the diagram, bear specific actions that support different stages of the SoS architectural evolution. Guidance on

information that supports each of these steps is as detailed in literature. The result is an establishment of a coherent flow of information throughout the SoS update cycle.

---

## **2.1 PROBLEM STATEMENT**

While the Wave Model establishes a concise framework on addressing SoS architectural evolutions, there is nevertheless a pressing need for adequate tools to support each stage of the decision epochs, and to navigate the complexities associated with decision making in such environments. Trades between capability and risk are essential during analysis of alternatives for SoS capability planning, namely in the evaluation of alternative architectures. Existing tools for such trades, where they exist, can be ineffective and non-intuitive when size and/or interdependency complexity is high. These features of interdependencies create a tradeoff space between development risks, cost and capability performance of a SoS, for which new tools/methods will need to be developed.

---

### **2.1.2 RESEARCH STRATEGY & RELATION TO SERC CORE COMPETENCIES**

The research relates directly to the SERC's core competency number 2 with its motivation from the challenges of systems engineering and acquisition when a system-of-systems capability is the primary objective. In these settings, systems must be able to interoperate and therefore have dependencies during the analysis of alternatives and concept development phases. It is well known that interoperability between systems may be necessary for SoS capabilities, but the underlying interdependencies in system development, if not addressed, conceal the impact of disruptions that cascade through the interdependence network. The tools to be derived from the methods research are in concert with several need areas identified in the "Trapeze Model" developed and described in the SoS SE Guidebook. In particular, the "Understanding Systems" element that seeks information about systems which impact SoS, both technical and programmatic, is addressed. Also, our approach for uncovering emergent outcomes in networked systems (resulting from interdependencies and topology features) is clearly congruent with the SERC's second Research Thrust in Systems Science and Complexity and also is congruent with the "Assess Performance" element in the "Wave" construct which seeks to identify potential undesirable emergent behaviors of the SoS and single system dependencies essential to SoS capabilities.

---

## **2.2 RESEARCH OBJECTIVES**

The objective of our effort is to develop and test methods that quantify the impact of system interdependencies and other architecture features in the context of SoS capability development, in support of the Wave Model. The products of the research can guide system engineering and architecture design activities. A variety of methods are investigated to conduct analysis of alternatives

while navigating the decision space that simultaneously considers the potential positive impacts of interdependencies (e.g., SoS capability) and negative impacts (e.g. consequences of disruption).

---

## 2.1 METHODS OF APPROACH

Our research is focused on a multi-pronged approach that utilizes a suite of tools; these computational tools leverage currently available methods, tailored specifically towards addressing the mixed nature of potential technical dimensions that practitioners may face in performing SoS evolutionary actions. The research conducted in this phase evaluates the applicability and ‘value added’ of each method in addressing potential SoS concerns. The methods and individual descriptions are as listed below:

- Robustness using Stand-In Redundancy  
The method employed here employs robust strategies in evaluating and constructing SoS networks of assets that have the ability to mitigate
- Interdependency and Resilience Analysis using Bayesian Networks  
This method utilizes statistical/distribution information from data driven SoS networks to address issues of network capability resilience and analysis of interdependencies in developmental networks.
- Interdependency analysis using Markov Networks  
This method employs Markov network chains to analyze the impact of interdependencies on development of interconnected networks of systems.
- Functional (Developmental) Dependency Network Analysis (F(D)DNA)  
This method extends the Markov network approach to analysis of both functional and developmental dependencies between constituent systems in a SoS architecture.
- Architectural development using Colored Petri Nets (CPN)  
This method employs a discrete event simulation technique to analyze event based architectures; discrete event networks are modeled as a combination of tokens, transitions and rule sets.
- SoS Architecture Decision Analysis using Robust Portfolio Optimization  
This method provides a decision support framework for the identification of ‘portfolios’ of interdependent systems that fulfill requirements and capabilities.

The methods and tools developed in this research, support: “*architecting, engineering and evolving SoS* and for designing and *engineering systems* which effectively support user needs as part of a larger SoS.” The first potential benefits lie in increasing the ability of system engineers to manage risk in the

presence of system interdependency complexities during the architecting of SoS capabilities. The distinctive traits of operational and managerial independence are key to making the collaboration work. The structure of the network of constituent systems, however, can contribute both negatively and positively to the successful achievement of SoS capabilities and developmental success. Risk in individual systems can explain the local impacts **but do not capture** the impact that disruptions to individual systems have at the enterprise level, where multiple systems – explicitly or implicitly interdependent – collaborate to achieve various capabilities.

In principle, the tools address all four types of SoS (*directed, acknowledged, collaborative, virtual*) since all types are sensitive to the cascading effects of disruption among interdependent systems. However, the *acknowledged* and *collaborative* are of primary interest in tailoring the methods because a) these include many DoD capability areas, b) these types exhibit managerial independence of systems and thus the likelihood of disruption propagation is high (due to potentially low awareness of dependencies). Further, one could argue that even *directed* SoS instances in DoD, if they are large enough, would benefit from greater awareness of cascading impacts of disruptions.

Insight provided from technical interrogation of decision engineering spaces and evolving the SoS relates back to the key activities that need to reside in the “Wave model” and the continual iteration of such activities in frequent epochs in SoS evolution and upgrade. Disruptions on the development of one system in a SoS can have unforeseen consequences on the development of others. Our research on the development and acquisition of SoS targets, within the context of the Wave Model addresses complexities stemming from system development risk, the interdependencies among systems, and the span-of-control of the systems or system-of-systems managers and engineers. The goal is to provide the necessary tools in making informed decisions on evaluating architecture alternatives, assessing risks and ultimately navigating the trades-offs between performance, cost and developmental risk.

### 3 ANALYTIC METHODS

---

---

#### 3.1 SoS RESILIENCE THROUGH STAND-IN REDUNDANCY

Resilience is the ability of a system or organization to react to and recover from disturbances with minimal effect on its dynamic stability (E. Hollnagel, Hampshire) . While the resilience of system-of systems (SoSs) depends on the reliability of their constituent systems, traditional reliability approaches cannot adequately quantify their resilience. Given the heterogeneity and often wide geographic distribution of SoS constituent systems, inclusion of backup redundant systems for a SoS is usually impractical and costly. In this work the impact of compensating for a loss of performance in one constituent system by re-tasking the remaining systems is quantitatively assessed. This is termed “stand-in redundancy”, and it provides a method to determine feasible alternative SoS configurations based on performance level recovery and cost of implementation.



---

### 3.1.1 INTRODUCTION

Traditional systems engineering practices try to anticipate and resist disruptions through classical reliability methods, such as inclusion of redundancy at the component level and use of preventive maintenance at the system level. Reliability analysis techniques, such as fault trees and event trees, are used to determine the level and types of redundancy to be included in the system design. Similar methods are used to develop maintenance plans to reduce the likelihood of failures at the system level. However, these approaches do not adequately satisfy the resilience needs of a SoS. Given the heterogeneity and, often wide geographic distribution, of the constituent systems, redundant systems in a SoS are impractical and costly. Additionally, high levels of interdependency between the systems imply increased risks of failures cascading throughout the SoS. These hurdles offer the opportunity to improve the resilience of the overarching system through unconventional means. This view echoes that of researchers who raise the need for a different perspective of resilience in the context of SoSs (Neches & Madni, 2012), (Sheard & Mostashari, 2008). A recent paper succinctly sums up the need for greater emphasis on resilient systems by stating that “systems should be made resilient, rather than merely reliable” as they “need to be able to recover from unexpected perturbations, disruptions, and degradations of the operational environment” (Madni & Jackson, 2009).

Here, a way to compensate for a loss of performance in one constituent system by re-tasking the remaining systems is studied. Specifically, as one entity, or node in a SoS, experiences degraded performance or a failure mode, other entities can alter their operations to compensate for this loss. This is called “stand-in redundancy, and it raises several interesting questions, such as: (1) given the failure of a specific system, what is the best configuration to compensate for the loss?; (2) what level of performance can be recovered with the new configuration?; and (3) what is the upstream effect of stand-in redundancy on development costs and risks?

To answer these questions, two concepts are developed: (1) reactive resilience, and (2) proactive resilience. *Reactive resilience* deals with performance recovery after a failure has occurred. In this case, for a specific capability, the reduction in overall SoS performance given various nodal failures is studied, and then the level of performance that can be recovered by reconfiguring the rest of the SoS is determined. Next, this idea is expanded to track the impact of gradual degradation of nodes. As the nodes degrade over time, the corresponding reduction in SoS performance could result in a situation where a different configuration might perform better than the current one. This implies that one might proactively transition the SoS to a different configuration before actual failure of the node. This is termed *proactive resilience*, as this transition to a new configuration before nodal failure occurs improves the robustness of the overall SoS. Figure 2 summarizes the above discussion and illustrates the different ways resilience in a SoS can be achieved. Failures can be addressed after they occur, through repair; or they can be anticipated and addressed before they occur, through preventive maintenance. Typically, exogenous methods are required to provide these services. In this work, the emphasis is on designing systems with the inherent ability to react to failures, without causing any downtime for either

maintenance and/or repair (as highlighted by the shaded region of the figure). This concept can be used to study various SoS architectures and evaluate their inherent ability to resist failures, thus in effect providing information for decision-makers to help identify “optimally” resilient SoS structures. Additionally, consideration of these resilience improvement techniques enables designers to better target risk resolution resources.

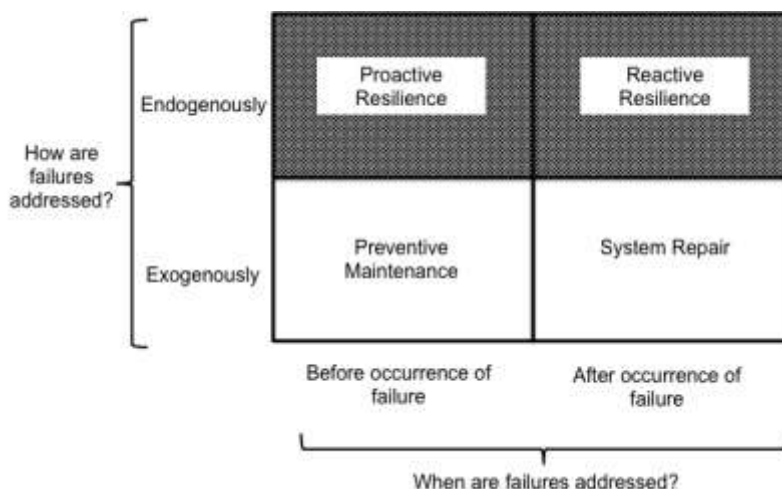


Figure 2. Classification of SoS resilience based on when and how failures are addressed

### 3.1.2 ANALYTIC FRAMEWORK

Consider a system-of-systems that consists of  $n$  systems. Typical representations of a SoS involve networked combinations of the constituent systems that ultimately provide SoS-level capabilities, as shown in Figure 3 (a). In this work, a SoS is viewed as a hierarchical structure of multi-system capabilities and single-system functions, as shown in Figure 3 (b). At the highest level, the SoS is essentially a collection of the capabilities that it is designed to provide. Subsequently, each capability emerges from a collection of system-level functions. At the lowest level, the functions are performed by the individual systems. For example, geosynchronous satellites can *image* large swathes of area for extended periods of time, and weapon-fitted UAVs can be used to *strike* targets in hostile environments. Consideration of these functions rather than the individual systems as the base level of SoS capabilities enables the resilience analysis of the larger system. As mentioned previously, the traditional practice of incorporating redundancy in complex engineered systems may not fit well with respect to evolving SoSs. Thus, this view of recapturing lost or deteriorating functions plays a key role in improving the resilience of SoSs. In the above example, if surveillance is a key requirement of the mission and if the satellite fails, then with appropriate imaging capabilities, the UAV can be re-tasked to perform surveillance functions and to network with systems that were originally linked to the satellite. At the middle level, groups of functions work together to provide a higher-level capability ( $C$ ). For example, the satellite could collaborate with a reconnaissance UAV to not only provide surveillance of a large area but also to provide high-definition imaging capabilities for target identification.

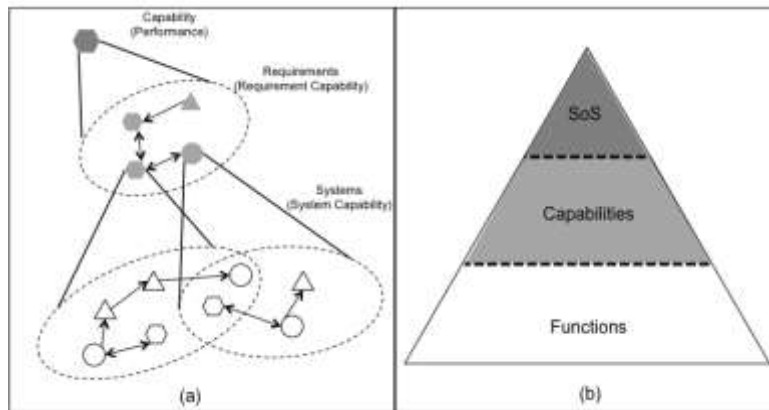


Figure 3. (a) Physical (Han, Marais, & DeLaurentis, 2012) and (b) functional representation of system-of-systems

Performance and reliability are two important metrics with respect to any operational system. Here, these two measures are considered at the capability level and are referred to as the Level of Performance (*LoP*) and Level of Reliability (*LoR*) respectively. Determination of the level of performance of SoS capabilities is challenging when compared to calculating these metrics for simpler systems. The level of performance achieved by a particular system function can be expressed by a direct measure of how well it performs its task. For instance, a direct measure of performance might be the area, say 1000 sq. mi., that a satellite can image with a given level of resolution. In contrast, determination of the level of performance for a particular capability is context and SoS dependent. Computing this metric relies on a number of factors such as the architecture of the constituent systems, the availability of these systems, the performance capabilities of each system, and the functions achievable by each system.

The level of reliability provided by a capability is relatively simpler to determine. There are well-established methods that help determine the reliability of systems (Rausand & Hoyland, 2004). Using these traditional methods and tabulated failure rate information, the reliability of a system ( $R_s(t)$ ), and by extension the reliability of the function it provides, at any time after it has been deployed can be computed. For this work, we define the level of reliability (*LoR*) of a capability as the probability that all the  $nc$  constituent systems contributing to that particular capability are operational at a particular time.

Figure 4 provides a graphical illustration of SoS operations using the metrics of Level of Performance (*LoP*) and Level of Reliability (*LoR*) described above. The desirable region of operation is in the top right of the graph, that is, the high-performance, high-reliability portion (as indicated by the blue circle). After the initial deployment, the systems gradually degrade with time, and the SoS region of operation moves to the left of the graph. In some cases, this degradation can result in a simultaneous reduction in performance levels as well as reliability (not shown in figure). If a system fails, the immediate loss in its functionality leads to a decrease in the overall performance level of the SoS (as shown by the red circle). Given the inherent characteristics of SoSs, a single system loss typically does not lead to a complete

failure of the larger SoS and hence, the overall LoP does not fall to zero. However, incorporating a certain level of stand-in redundancy will allow the SoS to minimize this performance loss without relying on external agents to either maintain or replace the failed system. In the event of a system failure, by allowing multiple systems in the SoS to perform the same functions, the remaining systems in the SoS can be re-tasked to perform the lost functions, even if to a lesser degree of performance. This is represented by a sequence of green circles, indicating that different levels of performance can be regained depending on the functional reconfigurability of the SoS

The framework to improve SoS resilience using stand-in redundancy is based on a combinatorial optimization approach. This method aims to choose the SoS configuration that minimizes the operational cost of the SoS, while achieving a certain level of capability performance, as well as a threshold level of reliability. For a SoS with  $m$  capabilities, the optimization formulation thus becomes:

Minimize:

*SoS Operations Cost*

Subject to:

$$LoP_i \geq LoP_i^T$$

$$LoR_i \geq LoR_i^T$$

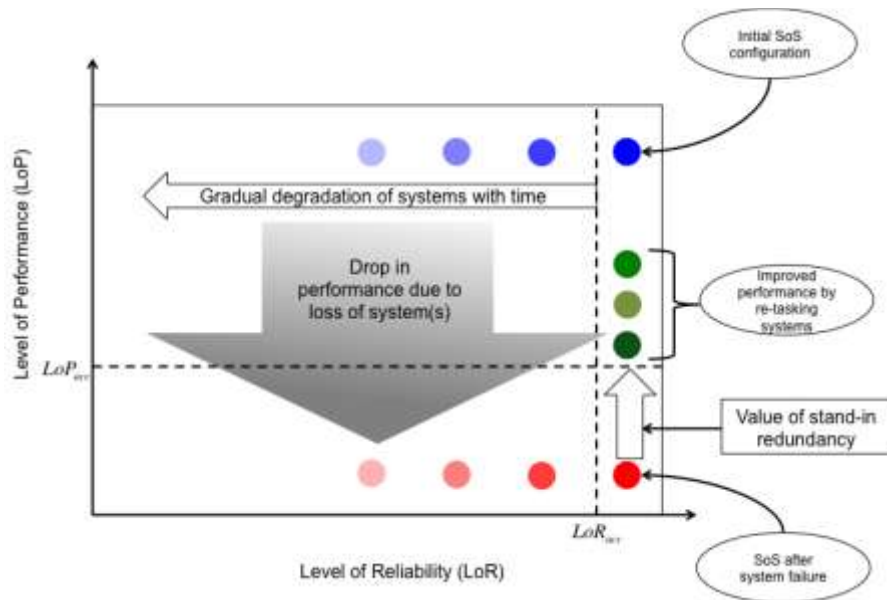


Figure 4. Notional LoP-LoR graph showing impact of stand-in redundancy on SoS performance

The SoS cost depends on a variety of factors and is highly context dependent. For this work, SoS operations is divided into three broad situations:

- Fully functional state: The SoS is in its original configuration with no system failures. The SoS cost depends on the number of systems contributing to the overall SoS capabilities as well as the cost of operating each system.
- System loss state: The SoS has suffered either single or multiple system failures and is now operating at a much lower level of performance. Here, the SoS cost depends on the operating costs of the remaining systems as well as costs that may be incurred in repairing and/or replacing the failed system.
- Re-tasked state: The remaining systems are re-tasked to recover some of the lost functionality after a system failure. In this situation, the SoS cost depends on the operating costs of the remaining systems, the acquisition costs to include additional features that enable this functional redundancy, and the marginal costs to re-configure the SoS so that systems can take on their new tasks

The constraints force the optimal solution to meet the target values of performance and reliability for each capability, defined by the decision-maker. Here, the target values are defined under two broad categories:

- Desired target values: These values are used to determine the region of operation (in the LoP-LoR graph) of the original, fully functional SoS. They represent the level of performance and reliability the SoS is expected to satisfy. For example, for a fully functional and newly deployed SoS, the systems can be chosen to provide relatively high target values for each capability.
- Acceptable target values: These values are used to determine the region of operation (in the LoP-LoR graph) of the re-tasked SoS. They represent the minimum acceptable level of performance that the remaining systems must provide. As these values are varied, the costs to achieve the corresponding value of stand-in redundancy, vary accordingly.

---

### 3.1.3 APPLICATION: AN ILLUSTRATIVE EXAMPLE

Consider the need for a SoS that provides the ability to detect and eliminate targets in hostile environments as well as provides large-scale surveillance in both hostile and non-hostile situations. The SoS consists of five systems: a geosynchronous satellite, three UAVs, and a ground station (see Figure 5). UAV-1 is primarily a surveillance drone with no weapons on board. On the other hand, UAV-2 and UAV-3 are fitted with weapons for target elimination tasks and they carry basic cameras on board to provide confirmation of the strikes that have been carried out. Combinations of these system functions yield three primary SoS capabilities: (1) surveillance, (2) target identification, and (3) target elimination. Using the representation developed in the previous, the SoS-level need is decomposed into capabilities as shown in Figure 5. For example, to provide surveillance capabilities, the corresponding systems need to be able to image large areas of land with high revisit rates. Figure 6 indicates the functions that each system can provide. With this setup, the systems that contribute to the SoS capabilities are:

1. Surveillance provided by the satellite
2. Target identification provided by a collaboration between the satellite and UAV-1
3. Target elimination provided by UAV-2 and UAV-3

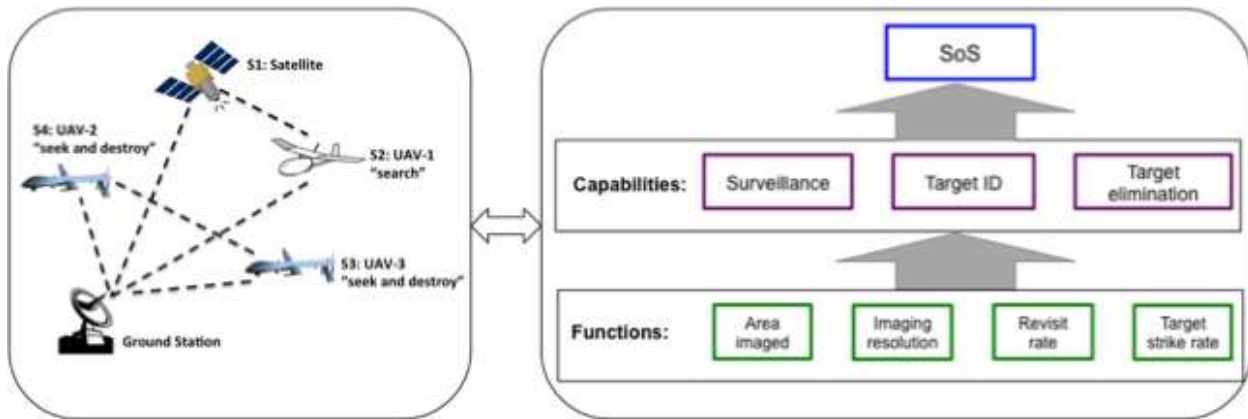


Figure 5. Five-system SoS and corresponding decomposition of capabilities

The main focus of this work was to assess the impact of incremental modifications to existing SoSs such that the overall architecture becomes more resilient to disruptions. Due to challenges related to its accessibility, the features on the satellite cannot be changed. In contrast, it is relatively easier to retrofit the UAVs with higher performance devices, such as sophisticated imaging and communication equipment. Additionally, UAVs may also be reprogrammed for higher revisit rates. The grey arrows in Figure 6 indicate the systems whose corresponding features can be changed

System ( $x_i$ )	Functions ( $fn_i$ )			
	Area imaged ( $fn_1$ )	Imaging resolution ( $fn_2$ )	Revisit rate ( $fn_3$ )	Target strike rate ( $fn_4$ )
Satellite ( $x_1$ )	✓	✓	✓	-
UAV-1 ( $x_2$ )	✓ ↑	✓ ↑	✓ ↑	-
UAV-2 ( $x_3$ )	✓ ↑	✓ ↑	✓ ↑	✓
UAV-3 ( $x_4$ )	✓ ↑	✓ ↑	✓ ↑	✓

✓ Indicates feature present in system in original SoS

↑ Indicates feature that can be upgraded/modified in system

Figure 6. Systems and the functions they provide

Next, the Level of Performance (LoP) and Level of Reliability (LoR) associated with each of the three capabilities are determined. Both these metrics depend on the needs of the particular SoS, the systems available to perform tasks, and the specific functions that can be provided by these systems. In this study, the LoP for each capability is assessed as the probability of achieving the particular capability. Given that combinations of system-level functions result in the corresponding capability, let  $S_{fn}$  denote the set of system functions that contribute to a capability ( $C_i$ ), and let  $S$  denote all possible combinations of these functions. Assuming binomial states of the systems, that is, each system is either

fully functional or completely degraded (failed), the probability of achieving a particular capability is determined by applying the law of total probability:

$$LoP(C_i) = \sum_S pr(C_i = 1 | S_{fn} = S) pr(S_{fn} = S) \quad (1)$$

This means that the probability of achieving a certain capability depends on (a) the conditional probability that the capability is achievable given the operational status of the systems and the functions they provide, as well as (b) the probability that the systems, and by extension, their functions, are operational. The latter probabilities can be determined based on the reliability of the systems at the time of interest, as below. On the other hand, calculation of the conditional probability of success is relatively more complex since it depends on a variety of factors, such as the actual number of operational systems, the strength of the links between the systems, and the availability of sub-systems within the systems. The reliabilities of each of these systems can be calculated by applying historical failure rate information:

$$LoR(C_i) = pr(\text{all constituent systems are operational at time } t) = \prod_{j=1}^{nc} R_j(t) \quad (2)$$

The optimization to determine the least costly configuration is performed under each of the following scenarios:

1. When nothing fails, that is, the original SoS performs as designed.
2. When one system fails.
3. When one or more systems are used to recover functionality after a system has failed.

The results are shown in Figure 7. The horizontal axis denotes the system that has failed; the vertical axis represents the LoP for the capability under interest. For each failed system, the first bar indicates the original fully functional SoS (the baseline case for further comparisons), the second bar represents the impact of the system failure on a particular capability, and the last bar shows the effect of re-tasking the remaining systems in the SoS to achieve a minimum acceptable level of performance. For example, consider  $C_1$  (surveillance) in Figure 7 (a). For this capability, the satellite alone provides the maximum LoP (100%) in the original SoS. If, however, the satellite fails, UAV-1 (with high definition imaging capabilities) can provide some surveillance capability (55%). On the other hand, if the systems were retrofitted with better imaging devices and reprogrammed for increased revisit rates, a combination of UAV-1 and UAV-2 are used to “stand-in” for the satellite, providing a higher LoP (72.5%) for the same capability.

In the case of  $C_2$  (target identification) (see Figure 7 (b)) loss of either the satellite or UAV-1 has a significant impact on this capability. Additionally, given the reliance of  $C_3$  (target elimination) on the ability to accurately track a target, it is vital that drastic performance decrements, especially in urgent



hostile situations, do not occur. When the satellite fails, assuming improved imaging capabilities on UAV-2, it can collaborate with UAV-1 to provide a marginally higher LoP. On the other hand, if UAV-1 were to fail, providing better imaging equipment on board UAV-2 (despite its primary role as an attack drone), raises the LoP by a significant amount (from 56% to 76%).

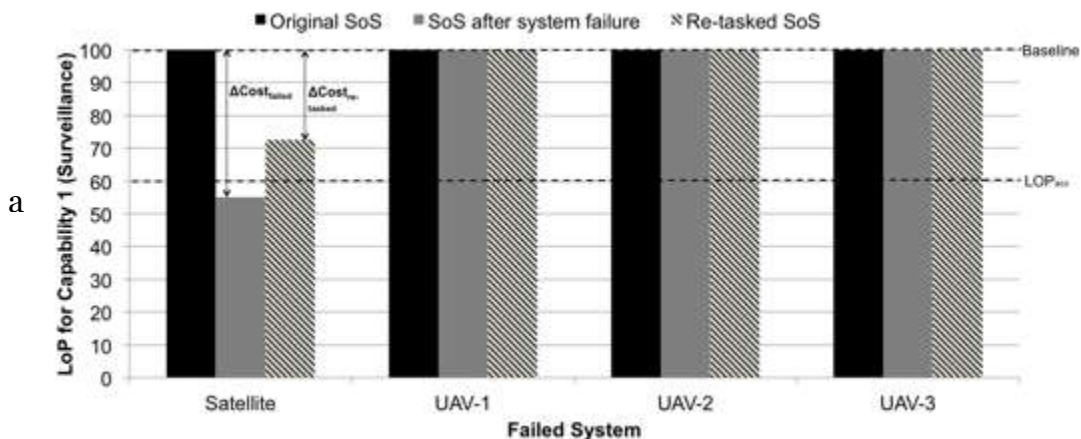
The impact of stand-in redundancy on  $C_3$  leads to some interesting observations (see Figure 7 (c)). While this capability directly stems from the targeting capabilities of the attack drones, namely UAV-2 and UAV-3, it relies heavily on accurate target identification information provided by  $C_2$ . Intuitively, the loss of either of the attack UAVs results in performance losses. For example, failure of UAV-2 alone leads to a direct loss of half the weapons striking capability of the SoS, and this loss cannot be recovered by any other system in the SoS as none of them are equipped to carry and launch weapons. In contrast, failure of either the satellite and/or UAV-1 adversely impacts the target identification ability of the SoS, thereby hindering its target elimination ability, despite the full functionality of UAVs 2 and 3. This highlights the fact that system failures can have significant impacts on capabilities that they are not directly designed to satisfy. In such situations, allowing other systems to take over some of the lost functionality helps maintain key capabilities at acceptable (as determined by decision-makers or operators) levels. For example, if UAV-1 fails in the midst of a raid, a properly equipped UAV-2 can provide target identification capability so that UAV-3 can carry out the actual target elimination tasks. Although this implies that UAV-2 may not be available to carry its own attack function, depending on the criticality of the situation, this marginal loss in firepower may be traded in for improved reconnaissance. As a result, it is important to keep in mind the immediate needs of the mission when deciding which configuration to transition to.

Cost considerations are highlighted in Figure 7 (a).  $\Delta Cost_{re-tasked}$  is the cost associated with re-tasking the remaining system and it depends on: (a) the features that are either modified or added to the existing systems (such as, inclusion of high performance cameras in UAV-1 and UAV-2), and (b) the operating costs of these systems. This metric is relatively easier to compute than the corresponding cost for the failed system,  $\Delta Cost_{failed}$ . Calculating this cost is challenging as it depends on: (a) the operating costs of the systems remaining in the SoS after a nodal failure, (b) the costs to replace the failed system (for example, deploy a new UAV to replace the failed one), as well as (c) the costs accrued in the downtime between system failure and system replacement.

These results indicate that resilience in SoSs can be improved without having to use traditional practices of backing up systems. Instead, systems can be designed to contribute to SoS-level capabilities in the ideal case, and to “stand-in” for failed functions in the event of a failure. It is also important to note that there is a limit to the level of stand-in redundancy that can be incorporated in such systems. Returning to the notional example used in this study, re-tasking of some systems is possible because the UAVs can be retrofitted with better cameras and imaging mechanisms, and/or can be reprogrammed for higher revisit rates. On the other hand, UAVs used for purely surveillance purposes cannot be easily fitted with weapons deploying capabilities, nor can the designed imaging capabilities of the satellite be changed. This further leads to interesting implications regarding the balance between improving the resilience of



an SoS, the costs associated with these improvements, and the need to incorporate features that not only improve the resilience but also ensure performance levels as the context of operations of the SoS changes with time. This is especially true of large-scale SoSs, such as multi-modal transportation networks, that are designed for long lifetimes with modifications and upgrades being incorporated in a gradual manner. Stand-in redundancy has the potential to improve the resilience of these systems, however, features and technological modifications that bring about stand-in redundancy need to be chosen keeping in mind the trade-offs between costs, resilience in face of current and future threats, and adaptability of the SoS in an uncertain future.



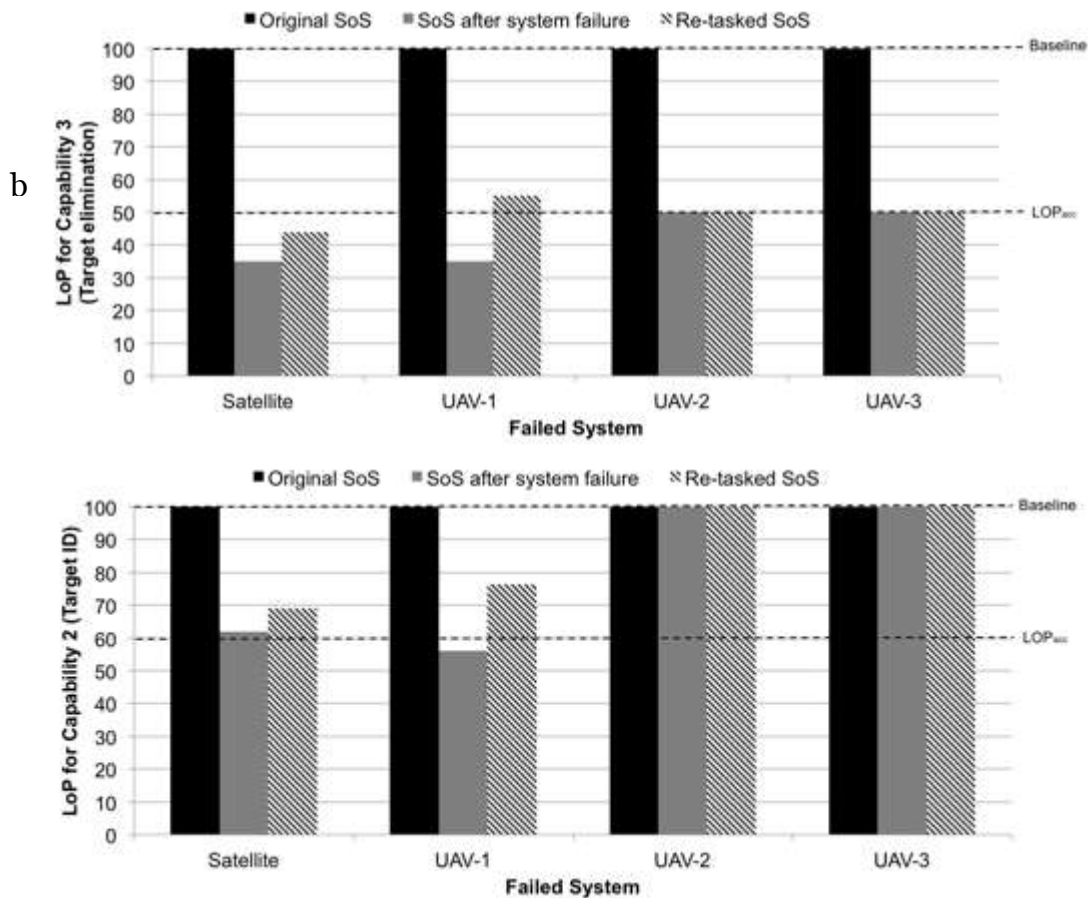


Figure 7. Impact of system failure and stand-in redundancy on (a) LoP of Capability 1 (Surveillance), (b) LoP of Capability 2 (Target identification), and (c) LoP of Capability 3 (Target elimination)

### 3.1.4 CONCLUSION AND FUTURE WORK

Traditionally, systems have been designed to be resilient through over-design. The emergence of large-scale system-of-systems (SoSs) has made it relatively hard to incorporate resilience in this manner as the system itself evolves with time along with its changing environment. This approach indicates that incremental enhancements and/or modifications to existing systems in these SoSs can provide inherent resilience. The concept of stand-in redundancy, allows cost-effective re-tasking and reconfiguration of SoSs. The resulting resilience capability minimizes performance loss at the SoS level in the event of an unanticipated system failure, and thus, enables improved operability of the SoS through uncertain futures. The next step of this study includes expanding this static model to a dynamic one with the use of stochastic tools to design for resilience under uncertainty. Additionally, the degradation of systems over time will be tracked in order to assess whether it might be beneficial to force a transition to a different configuration before actual failure of a system (proactive resilience). While the scope of this study was limited to single system failures, ongoing research aims at investigating the implications of stand-in redundancy in the case of multi-system failures. These concepts, and their resulting upstream

effects on development costs and risks, can be used by decision-makers to quantitatively assess the impact on resilience of different SoS architectures and their inherent ability to resist failures throughout the SoS lifecycle.

---

## **3.2 EVALUATING SoS RESILIENCE USING INTERDEPENDENCY ANALYSIS**

Interdependency between SoS systems, while enabling new capability, also means that failures can cascade throughout the SoS, creating development delays or additional system failures. Here we develop a method based on Bayesian networks to evaluate the resilience of SoS design alternatives to failures during operations. When designing SoS, engineers and decision makers are interested in identifying patterns of performance in response to failures in component systems. A metric that quantifies the capacity of a system to withstand failures can aid in design and operations choices. Therefore we define SoS resilience as the level of performance achieved relative to different levels of failure.

---

### **3.2.1 INTRODUCTION**

An SoS is a collection of distributed independent individual systems that interact with one another to achieve an SoS capability requirement that cannot be achieved by individual systems alone (in contrast, in a monolithic system, hardware or software components are integrated to form a single entity). Thus SoS are characterized both by independence and by interdependency. This interdependency, while allowing SoS to achieve their objectives, also means that failures can cascade throughout the SoS, creating development delays and/or operational system failures. For example, in the ballistic missile defense system, if the integrated sensor systems fail to detect a ballistic missile, then the interceptor system will also fail. The Joint Strike Fighter (JSF, or F-35) continues to face schedule and cost overruns, partially due to the cascading effects of component systems failing to meet requirements. Therefore, assessing SoS resilience to failures during development or operations requires considering cascading failures. These properties of independence and interdependence make evaluating SoS designs challenging. New approaches are needed to evaluate the risks associated with cascading system failures in SoS, both during development and during operation.

The objective in this section is to develop a method to evaluate the resilience of SoS design alternatives in the face of failures during operations. Interdependency analysis during development will be addressed in the next section (section 3.3). The research objective leads to three questions, which we address in this section:

- (1) How can we represent systems for resilience analysis?

In section 3.2.2, we briefly review the research on hierarchical representations and apply a hierarchical network representation to decompose SoS into three levels: capability, requirements, and systems.

(2) How can we analyze system interdependencies?

Section 3.2.3 presents our proposed method for performing interdependency analysis of SoS. We use a Bayesian Network model to analyze system interdependencies and to address uncertainty. This model captures the patterns of how system failures propagate to other dependent systems. These patterns allow us not only to identify critical subsystems that are responsible for potential system failures, but also to analyze SoS design alternatives.

(3) How can we measure system resilience?

In sections 3.2.4 and 3.2.5, we define metrics for assessing resilience. Then, we identify several patterns of resilience for different SoS architectures.

---

### 3.2.2 HIERARCHICAL ORGANIZATION

Hierarchical organization, which is a common feature of SoS, often leads to increased complexity when designing SoS due to three properties of components involved in SoS: the heterogeneity of constituent systems, the distributed nature of these systems, and the uncertainty in exploring the future state of the SoS (DeLaurentis & Callaway, Nov. 2004). Rasmussen proposed a means of representing complex systems to assist designers in building better interfaces, arguing that if system representation were more compatible with the operator's mental processes, there was greater likelihood of reducing human error and improving overall system performance (Goodstein, Andersen, & Olsen, 1988). DeLaurentis and Callaway proposed the SoS lexicon to define the explicit relative hierarchical positions of the different components according to their function and to categorize their roles in affecting a solution (DeLaurentis & Callaway, Nov. 2004). Based on the lexicon developed by DeLaurentis and Callaway, Ayyalasomayajula developed a framework for analyzing a SoS capability (Ayyalasomayajula, May 2011). In this research, we use the hierarchical representation as the backbone for analyzing systems interdependencies within the SoS.

---

### 3.2.3 AN INTERDEPENDENCY ANALYSIS OF A SYSTEM-OF-SYSTEMS USING A BAYESIAN NETWORK

We use a Bayesian Network (BN) to analyze uncertain interdependencies between systems. A BN is a probabilistic tool that evaluates networks of systems and graphically represents interactions among multiple components (Friedman & Koller, July 2009). It is a directed acyclic graph (DAG) whose nodes are the random variables and whose edges directly influence one another. Local probabilities represent the nature of the dependence of each variable (node) on its parents. Probability information in a Bayesian Network model is defined through these local distributions. A node with no parent node in the Bayesian Network model denotes a random variable and its associated probability distribution. A node with its parent node(s) can be represented as a conditional probability distribution.

Here, we use BNs to provide a basic structure for analyzing and visualizing the impact of failures during the operation of an SoS.

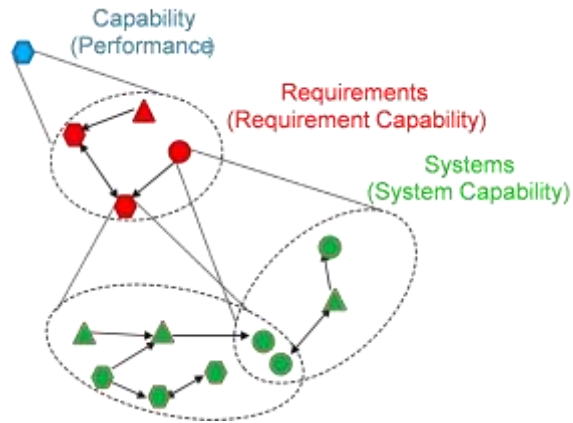


Figure 8. A hierarchical representation of a system-of-systems capability

Figure 8 shows a hierarchical abstraction of the desired SoS capability, the requirements that must be met to achieve this capability, and the physical systems that meet these requirements. In this setting, decision makers first define an intended mission and a desired capability in developing the SoS. Then, requirements are expressed in terms of capabilities and interdependencies between requirements. Systems are interconnected with each other to generate a capability to achieve requirements. This interdependency analysis provides the means to estimate the expected capability of the SoS when considering propagating effects between interdependent systems.

We estimate the expected capability by combining the probability of meeting each requirement with the capability provided if the requirement is met. Interdependencies between the requirements, and between the constituent systems, make this combination non-trivial. Consider the simple  $(n+1)$  node system shown in . The figure reads as follows: Node  $R$  has  $N$  parent nodes. Node  $R$  represents the requirement, and the parent nodes  $S_i$  represent the systems that meet this requirement. For example, in a ballistic missile defense system, one of the requirements may be the detection of a ballistic missile. To achieve this requirement, several systems such as unmanned aerial vehicles (UAV), satellites, radars, and receivers are needed.

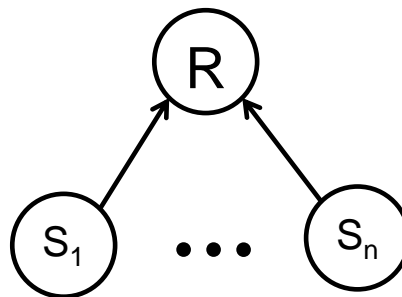


Figure 9. A Bayesian Network representation

Each node has its own failure rate. We consider here only complete success or complete failure, therefore successes and failures can be represented with a binomial distribution or derivatives thereof.

Let  $PA(R_i)$  denote the set of parents of the node  $R_i$ , i.e.  $\{S_1...S_n\}$ . Applying the law of total probability, the probability of achieving a particular requirement, node  $R_i$ , is:

$$p(R_i = 1) = \sum_{\mathbf{S}} p(R_i = 1 | PA(R_i) = \mathbf{S}) p(PA(R_i) = \mathbf{S}) \quad (3)$$

where  $\mathbf{S}$  is the set of all the possible combinations of parent node values. For example, if  $PA(R_i)$  includes two parent nodes  $S_1$  and  $S_2$ , then  $\mathbf{S} = \{(0, 0), (0, 1), (1, 0), (1, 1)\}$ . The conditional probability of success depends on the parent nodes and on their relationship to the dependent node. Returning to the detection requirement, if we use both a UAV and a satellite to detect an enemy, there are four possible combinations of these systems, with different combinations offering different probabilities of success.

In turn, the SoS capability depends on the combination of requirements. For example, consider an SoS with two requirements. The set of total combinations of requirements,  $\mathbf{R}$ , is  $\mathbf{R} = \{(0, 0), (1, 0), (0, 1), (1, 1)\}$ . When all requirements are met, i.e.,  $R = (1, 1)$ , then the SoS has full capability. However, if some of the requirements are not met, i.e.,  $\mathbf{R} = (1, 0), (0, 1)$ , or  $(0, 0)$ , then the SoS capability decreases. The probability of achieving the desired capability can be determined using the same approach as above. Finally, we combine all three levels to relate the capability to the system performance and reliability.

---

### 3.2.4 CONDITIONAL RESILIENCE IN THE SoS CONTEXT

Reliability is formally defined as the ability of a system to meet a given set of requirements for a given period of time in a given set of conditions. More recently, resilience has been proposed as a richer metric than reliability. The concept of resilience has been used to analyze systems and solve problems in fields such as ecology (Folke, et al., 2004), psychology (Masten, 2009), computer science (Mohammad, Hutchison, & Sterbenz, Nov. 2006), material science (Avallone, Baumeister, & Sadegh, 2007), and disaster management (Shinozuka & Chang, 2004). While a formal definition of resilience has not been agreed on as yet, resilience is generally understood to indicate the capacity of a system to survive, adapt and grow in the face of change and uncertainty (Fiksel, 2006).

Here, we attempt to apply these ideas to SoS. When designing SoS, engineers and decision makers are interested in identifying patterns of performance in response to failures in component systems. A metric that quantifies the capacity of a system to withstand failures can aid in design and operations choices. Therefore we define SoS resilience as the level of performance achieved relative to different levels of failure. Since failures in an SoS can occur in many ways, and have many different impacts, we propose two ways of measuring and communicating SoS resilience. First, we define the “conditional resilience” as the ratio of percentage of performance of SoS in response to failure in a particular system or combination of systems:

$$CR\left(\prod_i S_i\right) = \% \text{ of performance of SoS} \mid \left(\prod_i S_i\right) \quad (4)$$

where  $\prod_i S_i$  represents the particular combination of system failures.

The conditional resilience in this definition can be thought of as a ‘specific’ performance measure: it tells us how much performance is maintained for failure in a given set of systems.

### 3.2.5 TOTAL RESILIENCE IN THE SoS CONTEXT

The conditional resilience shows the system performance in response to a particular set of system failures. The total resilience pattern shows how performance is degraded as the total level of component system failures increases. Figure 10 shows a notional resilience pattern. The x-axis denotes the percent of systems that have failed and the y-axis denotes the performance (or capability) of the entire SoS. The yellow horizontal line represents the minimum acceptable level of performance to meet a system requirement. This value depends on the particular SoS being considered.

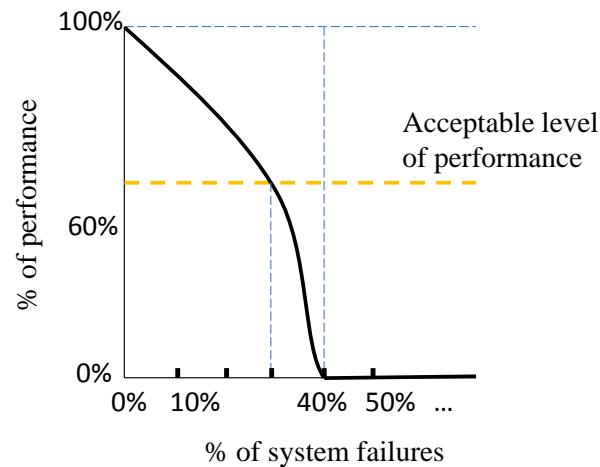


Figure 10. Performance-Failure diagram

Consider first a simple SoS where all the constituent systems are identical. Figure 11 shows three example resilience patterns with the same acceptable level of performance for this SoS.

- For a series system, the loss of one or more systems removes the entire capability. The resilience pattern is an impulse function.
- For a parallel non-redundant SoS of identical components without network effects, each loss reduces the SoS capability in the same way. The resilience pattern is a linear decreasing line.
- For a parallel system of identical components with network effects, there are two limit patterns. The red concave line shows the resilience pattern when the marginal performance benefit of each additional system is larger than the prior system. The convex green line shows the resilience pattern when the marginal performance benefit of each additional system is smaller than the prior system. A system following the red curve is brittle to small disturbances because small failures of a system sharply reduce the performance. However, a system following the green curve slowly reduces its performance with respect to failure levels.

The resilience patterns are influenced by two factors: architecture type and system-level risk of SoS. The architecture determines the general shape of the resilience pattern, as illustrated in the simple examples discussed earlier. The system-level risk determines the scale of the pattern. For the simple series and parallel SoS in the example, the resilience pattern can be derived quite easily. However, for

more complex SoS configurations, consisting of series and parallel combinations of heterogeneous systems, the pattern is harder to determine. The next section demonstrates an approach to determining the pattern for a Littoral Combat Ship system.

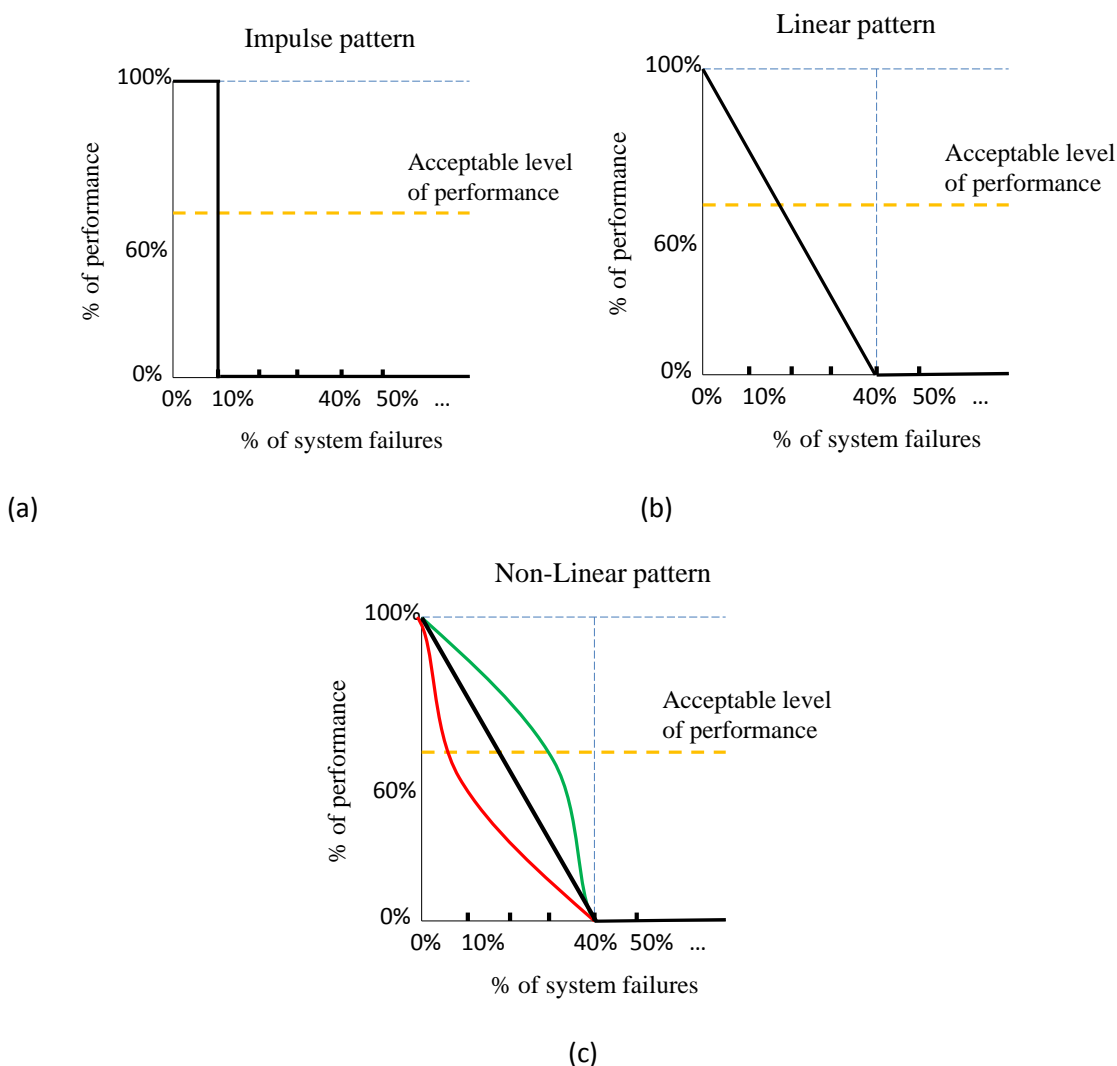


Figure 11. Three different resilience patterns

### 3.2.6 APPLICATION: A CASE STUDY OF LITTORAL COMBAT SHIP SYSTEM

The LCS is the first of a new family of surface ships for the US Navy. The LCS system is designed to counter growing potential threats in the littoral zone such as coastal mines, quiet diesel submarines, and terrorists on small, fast, and armed boats (O'Rourke, April 6, 2012), (LCS: The USA's Littoral Combat Ships, 2012). One of key features of the LCS system is the open architecture system, which allows for modular mission packages intended to facilitate flexible mission systems with quick upgrades and lower costs.



The LCS system is an SoS because it relies on several independent and interdependent constituents to achieve a mission that cannot be achieved by individual entities acting alone. The LCS SoS uses three different mission modules: mine warfare (MIW), anti-submarine warfare (ASW), and anti-surface warfare (SUW). It includes eight different systems: Armed Helicopter for Surveillance and Attack Missions (MH-60R), Helicopter for Airborne Mine Counter-Measure Missions (MH-60S), Unmanned Air Vehicle (UAV), Unmanned Surface Vehicle (USV), Remote Mine Hunting System (RMS), Torpedo, Missile, and the LCS. Since all systems are strongly dependent on each other to achieve missions, a system failure can cause mission failure. Here we demonstrate the application of our approach to compare two candidate LCS SoS architectures and to identify critical systems. In this application, we use two sets of LCS SoS to achieve missions as shown in Figure 12. For the first architecture, two LCS can communicate with each other, and for the second architecture, two LCS cannot communicate with each other. The presence (or absence) of the yellow dotted line classifies two different architectures.

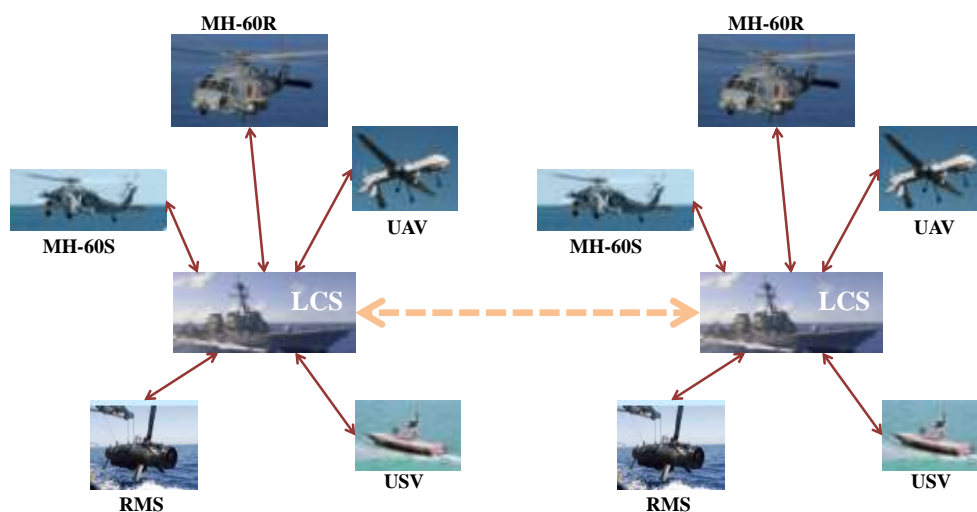


Figure 12. Overall architecture of the LCS SoS (LCS: The USA's Littoral Combat Ships, 2012), (Czapiewski, 2004)

The mission of the LCS SoS can be decomposed into three requirements: mine, anti-submarine, and anti-surface warfare. Figure 13 shows the decomposition of the LCS SoS requirements and systems. Each requirement is satisfied using several physical systems, and several systems contribute to more than one requirement.

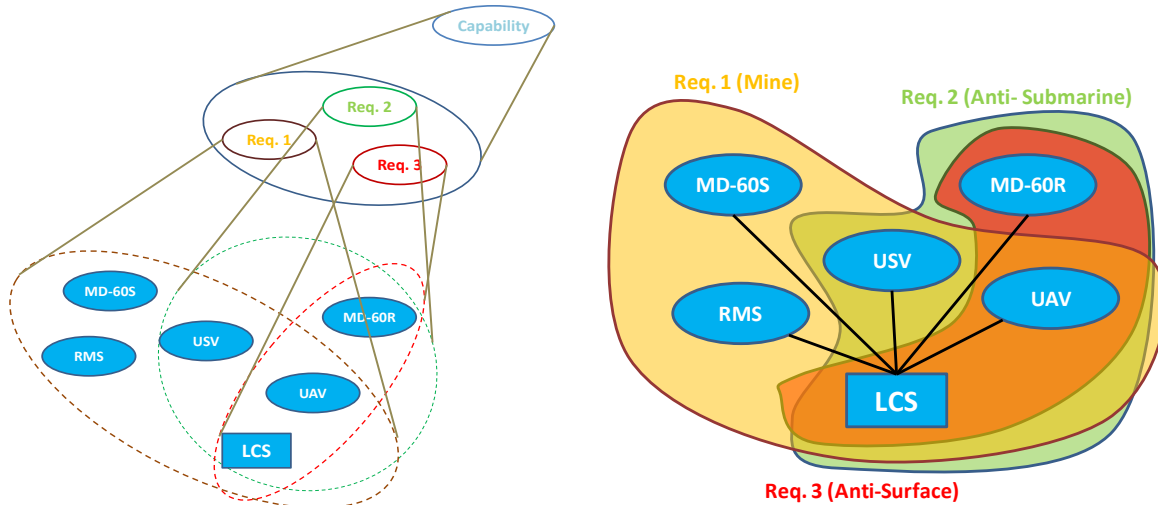


Figure 13. The decomposition of the LCS system

The first step in the interdependency analysis is to use a Bayesian network to determine the probabilities of successfully meeting the LCS SoS capability. These probabilities are a combination of the SoS design and the individual probabilities of meeting the requirements or of the constituent systems failing.

The overall capability of the LCS SoS is defined as the ability to complete a mission consisting of one or more of the three requirements. If we assume the requirements are equally important, then the probability of meeting the capability is given by:

$$p(C=1) = \sum_{\mathbf{R}} p(C=1 | PA(C) = \mathbf{R}) p(PA(C) = \mathbf{R}) \quad (5)$$

where  $\mathbf{R}$  is the set of all the possible combinations of the three requirements.

The next step is to determine the probabilities of meeting the requirements. To achieve each requirement, the LCS SoS performs two steps: (1) gathering information from observational systems to detect enemies, and (2) commanding a mission to perform the attack. For example, in the case of anti-surface warfare, two systems (UAV and MH-60R) gather information about enemy location. Then the LCS command center commands MH-60R and the missile-system in the LCS to perform the attack. Thus, the success of anti-surface warfare depends on the amount and quality of information on the enemy forces.

For example, if both the UAV and MH-60R are fully functional, then the LCS command center can gather 100% of enemies' information:

$$p(LCS=1 | UAV=1, MH-60R=1)=1.0 \quad (6)$$

If the MH-60R fails and the missile-system is working with 100% of enemy forces' information, then the LCS SoS can achieve 60% of the requirement 3:

$$p(\text{Req.}=1 \mid \text{UAV}=1, \text{MH-60R}=0, \text{LCS}=1)=0.6 \quad (7)$$

The final step in the interdependency analysis is to determine the system failure characteristics. These characteristics can be determined based on operational experience, which can for example be used to determine the mean-time-between-failure (MTBF). Alternatively, they can be estimated during design using techniques such as fault tree analysis. Here we assume that the failure times are exponentially distributed with notional MTBFs.

We consider here two candidate SoS architectures: (1) two LCS that can communicate with each other; and (2) two LCS that cannot communicate with each other. In the case of architecture 2, both command centers can use other resources for gathering and attacking enemies.

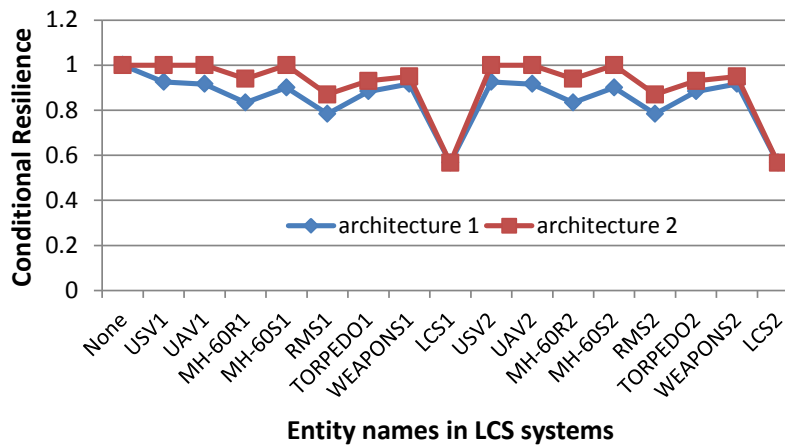


Figure 14. Conditional resilience values with one system failure in architecture 1 and 2

We identify the critical systems in each architecture using the conditional resilience metric provided in equation (4). The conditional resilience values are the ratio of percentage of overall capability of the LCS SoS. Figure 14 shows the conditional resilience values for individual system failures for the two architectures. The systems with the lower numbers are the critical systems. LCS1 and LCS2 are the most critical systems for both architectures, but in Architecture 1, overall conditional resilience values are less than those in Architecture 2.

Next, using the conditional resilience, we plot the resilience patterns as shown in Figure 15 for the two architectures. Since in this case, the constituent systems are heterogeneous, we first order the failures from most critical to least critical to determine the worst-case resilience pattern. Doing the converse yields the best-case pattern. We also order the systems from highest to lowest failure rate to give the expected-case pattern.

In this case the resilience pattern in the worst-case decreases sharply, but in the best-case it decreases gradually. The expected-case is in between the best and worst-cases. The results from Figure 15 support our earlier suggestion that the architecture type and system-level risk affect the SoS resilience. Further, in the presence of failures, Architecture 2 has higher performance levels than Architecture 1.

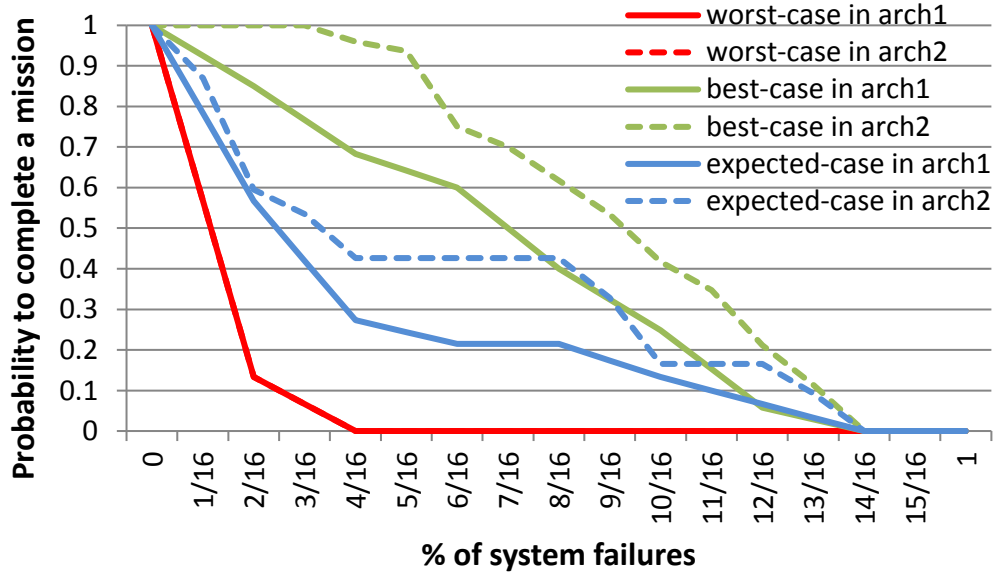


Figure 15. Resilience patterns with different failure rates of systems and different architectures

Finally, we determine the expected performance degradation over time by applying equations (5), (6), (7) and assuming that the System Probability Density Function (PDF) for each systems,  $i=1, \dots, 8$  is given by:

$$p(\text{System}_i = \text{Failure}) = 1 - e^{-\lambda_i t} \quad (8)$$

where  $\lambda_i$  is failure rate of system  $i$ .

Requirement conditional PDFs for  $k=1, \dots, 3$ :

$$p(R_k = 1) = \sum_S p(R_k = 1 | PA(R_k) = \mathbf{S}) p(PA(R_k) = \mathbf{S}) \quad (9)$$

where  $\mathbf{S}$  is the set of all the possible combinations of parent node values.

The capability conditional PDF is then given by:

$$p(C = 1) = \sum_{\mathbf{R}} p(C = 1 | PA(C) = \mathbf{R}) p(PA(C) = \mathbf{R}) \quad (10)$$

where  $\mathbf{R}$  is the set of all the possible combinations of the three requirements.

The exponential distribution is a function of time; the reliability decreases with time. This makes sense in the LCS example. As the exposure time of a system in the LCS set increases, the reliability of the system decreases. Figure 16 shows the evolution of two architectures' performance. The result again demonstrates the sensitivity of performance to architecture type. Architecture 2 is more resilient than Architecture 1. The source of this increased resilience is the communication connection between the two LCS, which allows the two LCS systems to share their resources.

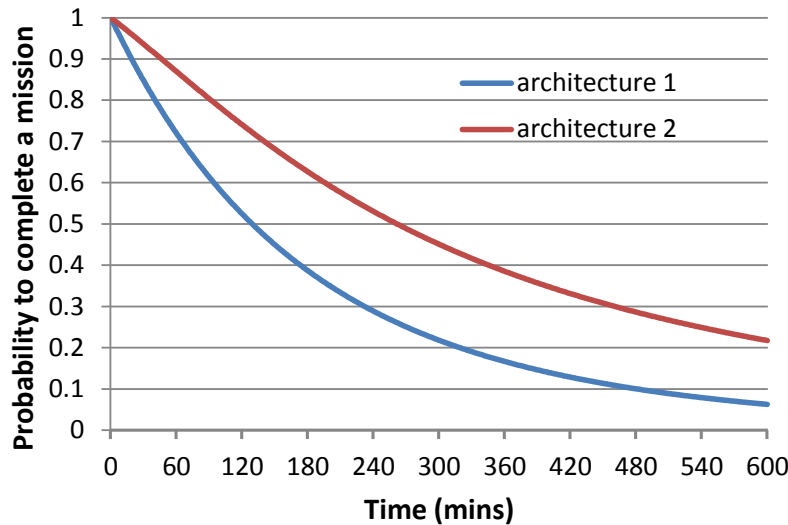


Figure 16. Evolution of the LCS SoS performances of both architectures for 600 minutes

### 3.2.7 CONCLUSION AND FUTURE WORK

In this section, we proposed ways of representing and evaluating SoS resilience, using Bayesian networks. We proposed a conditional resilience metric that measures each constituent system's contribution to overall SoS resilience, and a resilience pattern that shows how SoS performance degrades as systems fail. We demonstrated our approach using the Littoral Combat Ship example.

SoS resilience is determined by both the SoS architecture and the constituent system reliability. In the proposed example, we determined the two most critical systems using the conditional resilience metric. Adding a communications link between these two systems increased the resilience, resulting in higher expected performance and slower expected performance degradation as a result of system failure.

In future work we will further explore the application of the conditional resilience metric and the resilience pattern. In particular, while we showed resilience patterns for simple SoS configurations, more work is required to evaluate their application in more complex SoS configurations.

## 3.3 DEVELOPMENT INTERDEPENDENCY MODELING FOR SYSTEM-OF-SYSTEMS (SoS) : SoS MANAGEMENT STRATEGY

Event tree methods and Bayesian Networks (BNs) are used in this section to quantify development interdependencies between systems and assess cascading development risks. In addition the approach also allows inputs (e.g. development failure rates of participating systems) to be updated automatically for better decision-making. A primary output of the approach is the quantification of development interdependencies and the identification of critical systems with respect to propagating effect levels.

The outcomes of the analysis provide a support for decision makers to manage risk in development of a SoS with complex interdependencies.

---

### **3.3.1 INTRODUCTION**

Whilst giving a SoS the characteristic capability of achieving its objectives, this interdependency can also cause failures to cascade through the SoS thereby causing potential development delays. Therefore, these properties of independence and interdependence make project management of SoSs challenging during the development process. For example, the Joint Strike Fighter (JSF, or F-35) continues to face schedule and cost overruns, partially due to the cascading effects of component systems failing to meet requirements. Many surveys show the schedule and cost overruns are serious problems in a SoS, especially defense programs. The Government Accountability Office (GAO) report estimated that the 98 Major Defense Acquisition Programs (MDAPs) from FY2010 collectively overran their schedule by an average of 22 months and budgets by \$402 billion (Dodaro, 2010). According to an independent report by the British Ministry of Defense, defense programs overran their schedule by an average of 40% leading to overall cost increases by 40% (Gray, 2009). Therefore, an adequate assessment of the cascading effects of risk among interdependent systems during the development process has the potential to reduce cost and schedule overruns. The rationale behind this is that such analyses can reduce risk of a project management.

Several approaches have been developed to analyze systems interdependencies during operations and associated propagating impacts in different domains (Xu, Donohue, Laskey, & Chen, June 2005), (Han & DeLaurentis, Sep. 2011), (Nishijima, Maes, Goyet, & Faber, March 26-27, 2007). These studies focus on analyzing operational interdependencies between systems using data that is more readily available than the data required for analyzing development dependencies. In such scenarios researchers are restricted to carrying out analysis with limited information such as sparse data or expert opinions, and are hence struggling to quantify development dependencies. Therefore new approaches are needed not only to quantify the development dependency strengths with cascading system failures in a SoS during the system development but also, to deal with uncertainty.

We propose a method to quantify the development dependency strengths between system builders. The method employs Bayesian Networks (BNs) to represent interdependencies between system builders in a SoS capability development context. The BN uses two inputs: 1) development failure rates of system builders and 2) dependency strength between system builders. BNs are particularly suited to such problems given their efficacy for representing causal relationships between systems involving uncertainty. Uncertainty is represented in a BN using beta distributions which increases the robustness of model outcomes. The BN is used on a synthetic problem to compute the impact of interdependencies between system builders. Results are described in the context of their use by decision makers to manage risk in development of a SoS with complex interdependencies.

---

### 3.3.2 OVERVIEW OF DEVELOPMENT INTERDEPENDENCY MODELING FOR SoS

There are two ways to develop a new SoS: 1) integrate only existing off-the-shelf systems or 2) deploy nascent and inchoate systems with existing ones. In many cases, the latter is selected for the sake of the advancement of SoS wide capability. It is important for project managers to effectively estimate possibilities of developing new systems. However, estimating development possibilities for all activities is a challenge due to the complex interdependencies between systems activities organized for a SoS capability. We use failure rates of systems activities to represent possibility levels for a system development. Once a failure rate of each activity (in consideration with its interdependent activities) has been estimated it can be used for better decision making to reduce schedule and budget overruns. We develop an integrated simulation model to estimate failure rates of system activities in a SoS. The model identifies the system which is most susceptible to disruption propagation impacts. The outputs from the integrated simulation model are the estimated failure rates of all activities at scheduled time. The integrated simulation model is comprised of five principal steps (Figure 17) as described by the following:

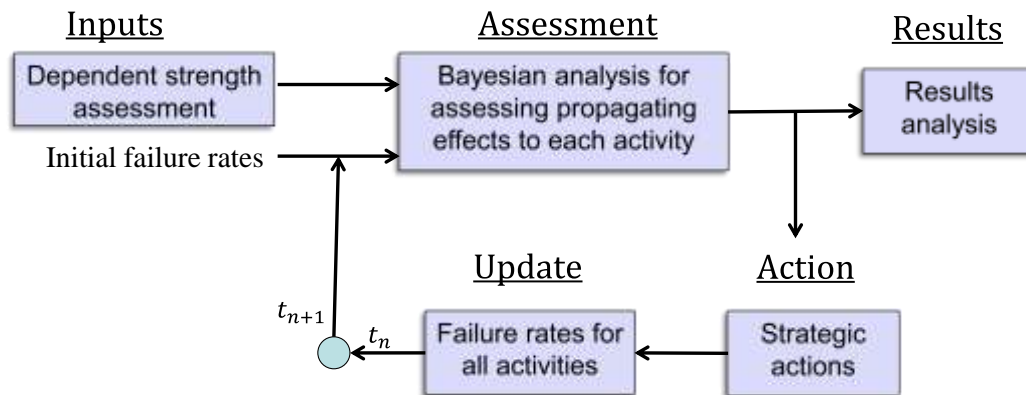


Figure 17. Overview of the integrated simulation model for development interdependency analysis

---

### 3.3.3 INPUTS FOR DEVELOPMENT INTERDEPENDENCY MODELING – INITIAL DEVELOPMENT FAILURE RATES AND DEPENDENCY STRENGTH

The first step in the modeling development interdependency analysis is to estimate initial development failure rates and the dependency strength of activities. Initial development failure rates of system activities depend on technology maturity levels and the system builder's potential for the new system development. If a system builder has mature technologies or high potential to develop a new system, then initial development failure rate for that activity is low. It is important to account for uncertainty in the model and data in order to generate reliable results. Hence beta distributions are used to represent initial development failure rates to address uncertainties. Beta distributions are a family of continuous probability distributions defined on the interval between 0 and 1, parameterized by two positive shape parameters ( $\alpha$  and  $\beta$ ). There are several reasons for justifying the use of beta distributions in the

proposed model. First, beta distribution allows for the representation of various types of probability information (Reese, Johnson, Hama, & Wilson, 2005). Figure 18 presents various shapes of beta distributions according to two positive shape parameters. Second, when probability information is unavailable, the beta family is the best choice for determining the most conservative probability information (Dyer & Chiou, 1984).

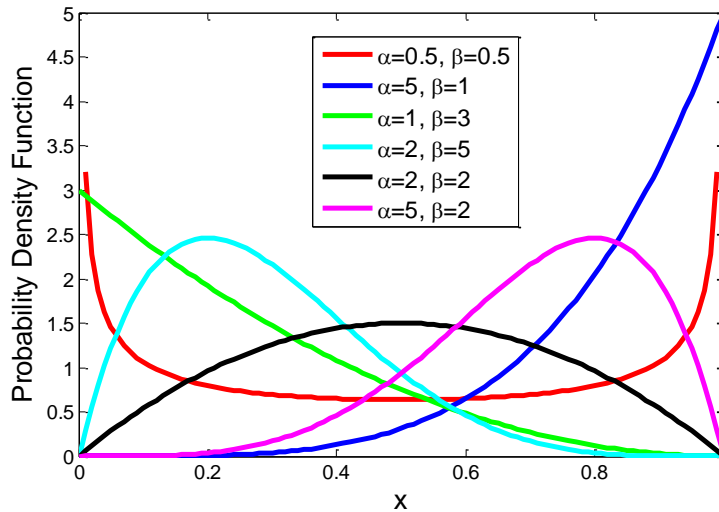


Figure 18. Various shapes of beta distributions

The second major input, dependency strength, plays an important role in analysing propagating effects among interconnected systems. We use *requirement connectivity matrix* and *system maturity levels* to quantify dependency strength between system activities. In this study, dependency refers to requirements dependency. We assume that the individual component systems in our SoS are built independently. If there does not occur any problems during system development, development of the component systems should not have propagation of disruption between each other. However, if a disruption of the system development happens, then this disruption may propagate. For example, Unmanned Aerial Vehicle (UAV) and a satellite are constituent systems of a SoS. One of the SoS requirements is to send/receive data with high bandwidth between UAV and the satellite. To achieve this requirement, system builders should communicate with each other to develop compatible data transmission systems. This dependency strength is called '*requirement dependency*'. The dependency strength represents the chance that disruption in the activities of the UAV builder will propagate to the Satellite builder. The detailed process to estimate requirement dependency strength is as follows.

### 3.3.3.1 Connectivity Matrix for Requirement Dependency

The connectivity matrix is a column matrix indicating which requirements of a system depend on the other systems. Suppose in our last example, the UAV and the Satellite have three requirements each and one pair of requirements is dependent. If UAV's 3rd requirement is dependent on the Satellite's 1st requirement, then connectivity matrix for UAV and the Satellite can be represented by  $Con_{UAV \rightarrow Sat} = [0 \ 0$



$1]^T$  and  $\text{Con}_{\text{Sat} \rightarrow \text{UAV}} = [1 \ 0 \ 0]^T$  respectively. The connectivity matrix only allows binary entries 0 or 1, where '0' represents independence and '1' represents independence.

### 3.3.3.2 Conditional Probability of a Requirement Failure Given a System Builder Failure

The second step is to quantify requirement dependency strength through estimating conditional probabilities of requirement failures using technology maturity scales. In the development process for high technology systems, system builders may not deliver new systems on time due to lack of technology maturity. In this study, we use a system-focused prescriptive metric entitled System Readiness Level (SRL) to estimate the technology maturity scales of a requirement. A requirement can be hierarchically decomposed into several systems needed to achieve the requirement capability. Figure 19 represents the hierarchical representation of requirements and the capability of an entire system.

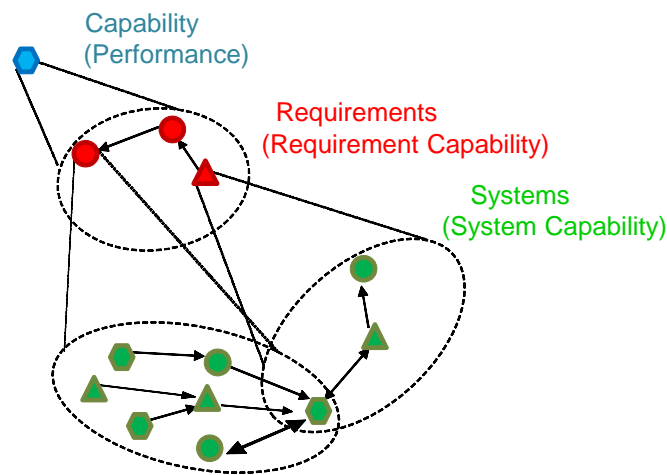


Figure 19. Hierarchical representation of an entire system

SRL is defined as a function of Technology Readiness Levels (TRLs) of component systems and Integration Readiness Levels (IRLs) of the integrations. TRLs are a systematic metric/measurement, invented by NASA that supports assessments of the maturity of a particular technology and the consistent comparison of maturity between different types of technology (Mankins, 1995). IRLs provide an integration specific metric to determine the integration maturity between two or more component systems (Gove, Sauser, & Ramirez-Marquez, 2007). Requirements can be interpreted as systems with several component systems. Therefore we can estimate the technology maturity scales for requirements using SRL metrics. In order to address uncertainty into SRL metrics, Tan, et. al. (Tan, Ramirez-Marquez, & Sauser, 2011) developed a probabilistic approach which provides a SRL probability distribution incorporated information of relative frequency of the TRL/IRL values provided by system evaluators. SRL can take values between 0 and 1. A system with a higher SRL value represents a matured system. The matured system has a smaller residual of the SRL value, unity minus a SRL value. We use mean values of SRL probability distributions to calculate the conditional probability of a requirement failure by normalizing residuals of SRL values for all requirements. If three requirements of UAV have SRL values of

0.4, 0.8, and 0.5 respectively, the conditional probability of requirement 1 failure given the UAV builder failure is as follows.

$$P(UAV_{Req1} = F | UAV = F) = \frac{SRL\ residual\ of\ req_1}{\sum_{i=1}^3 SRL\ residual\ of\ req_i} = \frac{(1-0.4)}{(1-0.4)+(1-0.8)+(1-0.5)} = 0.46 \quad (11)$$

After applying this process to all requirements, we can obtain a matrix including conditional probabilities of all requirements,  $P(UAV_{Req}=Failure | UAV=Failure)=[0.46\ 0.15\ 0.39]$ .

### 3.3.3.3 Quantify Requirement Dependency Strengths

Requirement dependency strengths are obtained by multiplying the connectivity and conditional probability matrices. The dependency strength represents the probability that disruption of a system builder will propagate to its dependent system builder. The failure of UAV builder may propagate to the satellite builder with the probability of 0.39 as calculated in the following equation:

$$Dependency\ Strength(UAV \rightarrow Sat) = P(UAV_{Req}=F | UAV=F) \times Con(UAV \rightarrow Sat) = 0.39 \quad (12)$$

---

## 3.3.4 AN INTERDEPENDENCY ANALYSIS OF A SYSTEM-OF-SYSTEMS USING A BAYESIAN NETWORK

### 3.3.4.1 A Bayesian Network

As mentioned in the previous section, a Bayesian Network is a directed acyclic graph (DAG) whose nodes are random variables and whose edges directly influence one another. Local probabilities represent the nature of the dependence of each variable (node) on its parents. Probability information in a Bayesian Network model is defined through these local distributions. A node with no parent node in the Bayesian Network model denotes a random variable and its associated probability distribution. A node with its parent node(s) can be represented as a conditional probability distribution (CPD). The first important step to build the BN is to construct the network of interests while considering dependencies between nodes and to estimate failure rates of nodes. Requirement dependency strength and failure rates of system builders mentioned in section 3.3.3 are used to form the BN.

In this study, a Bayesian Network (BN) is adopted to analyze interdependencies between systems. The BN can graphically represent interactions among multiple components and provide the basic structure for analyzing and visualizing the development SoS model. The BN is a probabilistic tool that evaluates networks of systems with respect to disruption propagation in developing systems for a SoS. The evaluation not only identifies critical components from a risk perspective, but also can show disruption and dependency effects on total expected development time of the SoS capability.

### 3.3.4.2 Propagating System Failures through Interdependencies

There are two sources of system failure in a SoS context: inherent and propagating. If a system fails on its own, then it is called an inherent failure. However, if a system failure is caused by propagating effects from interdependent systems, it is then called a propagating failure. It is therefore important to fuse all failure information, inherent and propagated.

Figure 20 shows a simple Bayesian Network where the node Y has N parent nodes. The research here focuses on binomial failures for a node. For example, each node in the Bayesian Network can only take values 0 or 1 to represent the status of the component as ‘failure’ or ‘working’, respectively. This is a limiting factor of the approach which we will revisit at the conclusion.

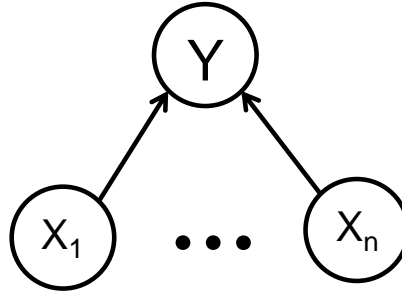


Figure 20. A Bayesian Network representation

Consider that each node has its own inherent failure rate defined by a beta distribution and node Y has n parents,  $X_1 \dots X_n$ . A Beta distribution is parameterized by two positive shape parameters, denoted by  $s_{n+1}$  and  $s_{n+n_n+1}$ . These two positive parameters are interpreted as the number of failures and survivors respectively when  $s_n$  and  $n_n$  are integers (Springer & Thompson, 1966). Let  $PA(Y)$  denote the set of the parents of the node Y, i.e.  $\{X_1 \dots X_n\}$ . According to the law of total probability, the propagating failure rate of node Y is:

$$p(Y = 0) = \sum_k CPD_k p(PA(Y) = k) \quad (13)$$

where  $CPD_k \equiv p(Y = 0 | PA(Y) = k)$ ,  $k$  is all combination of parent node values. For example, if  $PA(Y)$  includes two parent nodes ( $X_1$  and  $X_2$ ), then  $k = \{\{0,0\}, \{0,1\}, \{1,0\}, \{1,1\}\}$ . Therefore, a node with two parent nodes has four  $CPD_k$  values.  $CPD_k$  values here indicate the dependency strength of a failure propagating to a dependent system. For instance, if node  $X_1$  (or  $X_2$ ) fails, then node Y has 30% (or 20%) of chance to fail by a propagating effect of the node  $X_1$  (or  $X_2$ ) failure. In this case, all  $CPD_k$  values are determined:

$$p(Y = 0 | X_1 = 1, X_2 = 0) = 0.3, \quad p(Y = 0 | X_1 = 0, X_2 = 1) = 0.2, \\ p(Y = 0 | X_1 = 0, X_2 = 0) = 0, \text{ and } p(Y = 0 | X_1 = 1, X_2 = 1) = 0.5.$$

Analytical solution,  $p(Y = 0)$ , for the propagating failure rate of node Y is very likely to be non-parametric due to its complicated functional form. For computational convenience, we use the approach in reference (Liu, Li, & Kim, 2011), (Thompson & Haynes, 1980) to approximate the propagating failure rate with a beta distribution having the same first two moments. Let  $\text{beta}(b,c)$  denote the beta distribution of the propagating failure rate of node Y. Then, the first two moments of this distribution are:

$$M_1 = E(Y) = \frac{b}{b+c}, \text{ and } M_2 = E(Y^2) = \frac{b+1}{b+c+1} E(Y) \quad (14)$$

The first moment of  $p(Y=0)$  is the mean:

$$M_1 = E(p(Y=0)) = E\left(\sum_k CPD_k p(PA(Y)=k)\right) = \sum_k CPD_k \prod_i E[p(PA_i(Y)=j)] \quad (15)$$

where  $j$  is the value for  $PA_i(Y)$  in the set of  $k$ . For computational ease, equation (15) can be further written as the follows.

$$M_1 = E(p(Y=0)) = \sum_k CPD_k \prod_i \{j_i - E[p(PA_i(Y)=0)]\} = \sum_k CPD_k \prod_i \left\{j_i - \frac{s_i+1}{n_i+2}\right\} \quad (16)$$

The second moment of  $p(Y=0)$  is the mean of  $p(Y=0)^2$ :

$$M_2 = E(p(Y=0)^2) = \sum_k CPD_k \prod_i \left\{j_i^2 - 2j_i \frac{s_i+1}{n_i+2} + \frac{(s_i+1)(s_i+2)}{(n_i+2)(n_i+3)}\right\} \quad (17)$$

Finally, we can define two parameters,  $b$  and  $c$ , for a beta distribution of  $p(Y=0)$  as

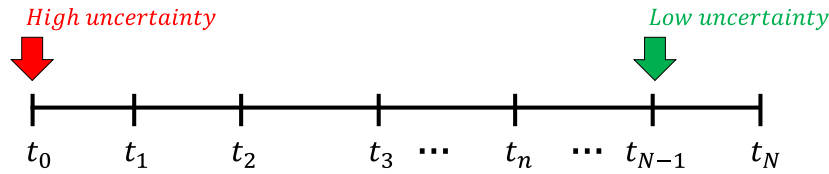
$$b = \frac{M_1(M_1 - M_2)}{(M_2 - M_1^2)}, \quad c = \frac{(1 - M_1)(M_1 - M_2)}{(M_2 - M_1^2)} \quad (18)$$

Now, node  $Y$  has two different beta distributions, one for inherent failure rate and one for propagating failure rates encapsulated in equation (18). These two beta distributions are integrated to get the new failure rate distribution of node  $Y$ . This task can be easily completed using the same process for obtaining fusion of all propagating failure information mentioned above, with different CPDs indicating 100% propagating effects. After applying these two fusion processes (the first being the fusion of propagating effects from dependent systems for the propagating failure rate and the second being the fusion of both inherent and propagating failure rates for the new integrated failure rate) to all nodes in a network, we can obtain the beta distributions of the new failure rate information including the propagating effects for all nodes. This result can be used to determine the critical (vulnerable) component and the expected development time for the SoS.

### 3.3.5 UPDATE OF FAILURE RATES FOR ALL ACTIVITIES

The failure rates of the system development may vary with time. If the system builder follows the schedule without any hiccups during the system development phase, development risk may decrease. In addition to evolution of the failure rates, uncertainty values in the failure rates of the systems decrease as time elapses. For example, assume that there are N times that the schedule is checked during a system development (Figure 21). At each time step, a project manager estimates the failure rates of the system. In the beginning of the system development, uncertainty in the failure rates is high. However, once he knows the status of the system development at the time  $t_n$ , he may estimate more accurate failure rates of building the system. Therefore, for an accurate and reliable result, the proposed method includes analysis of update of failure rates for all activities with uncertainty.

Figure 21. Uncertainty on the time steps



In this study, we use event tree analysis and beta distributions to update failure rates of all systems. An event tree is a commonly applied technique used for identifying the consequences following the occurrence of potential events such as failures or successes. It was first applied in risk assessments for the nuclear industry but is now utilized by other industries such as chemical processing, offshore oil and gas production, and transportation. We can estimate the outcome of the SoS development failure rate by quantifying the event tree diagram. Figure 22 shows a very simple event tree structure for the development schedule of a SoS. All events are the scheduling checks during the SoS development. The branch points consider the failure and success of each event. The outcomes determined by the end point of each event tree branch identify a different value for failure or success following the initiating event. Total failure rate of a SoS development can be obtained by summing up all outcomes of failures at the end points of the event tree. If we know the initial failure rate and conditional probability of the system failure given the system failure or success at the previous step, then the event tree quantification is the simple task of multiplying the probabilities of passing along each branch point on any path through the diagram by the conditional probabilities. For example, let the initial system failure rate and conditional probabilities of the system failure probabilities given failure and success be  $P(F_{t_0})=0.1$ ,  $P(F_{t_1}|F_{t_0})=0.4$ , and  $P(F_{t_1}|S_{t_0})=0.6$ . Then system failure rate  $P(F_{t_1})$  at time  $t_1$  can be calculated using the following equation. All failure rates at any time steps can be estimated in the same way.

$$P(F_{t_1}) = P(F_{t_0}) \times P(F_{t_1} | F_{t_0}) + \{1 - P(F_{t_0})\} \times P(F_{t_1} | S_{t_0}) = 0.1 \times 0.4 + 0.9 \times 0.6 = 0.56 \quad (19)$$

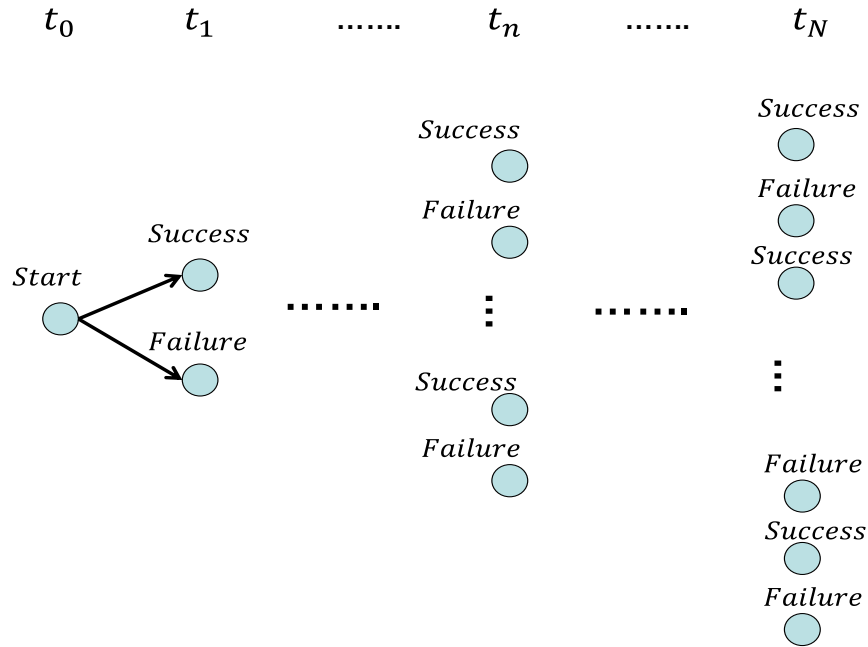


Figure 22. A very simple event tree structure for an example development schedule

The historical data is used to estimate the initial failure rate and conditional probability of the system failure. In the absence of such information, we can also use uninformative distributions as initial inputs such as, uniform distribution. As time elapses, we observe the results at each time step. These observations can be used to update conditional probabilities of the system failure by adding weighted numbers to parameters of beta distribution. This process allows uncertainty of the system failure to decrease as time goes by.

### 3.3.6 APPLICATION: A SYNTHETIC DEMONSTRATION PROBLEM

A simple synthetic problem is formulated and solved to demonstrate the proposed approach. Figure 23 shows the representation of a five-system network, where, the development of system 1, here denoted by S1, depends on the development of system 2, 3, and 5. This implies that a failure from one system development process affects the development of dependent systems due to requirement interdependencies between systems. For system 1, a failure cascades from system 2, 3, and 5 to system 1. The T values indicate the requirement dependency strength and correspond to the conditional probability of a failure propagating to a dependent system. For instance, if system 3 fails, then system 1 has 25% of chance to fail by a propagating effect of the system 4 failure. In order to estimate the requirement dependency strengths for all pair of systems, all systems should be decomposed into requirements and further into constituent systems. Then connectivity matrix and system readiness levels (SRLs) can be obtained using requirement relationships and TRL/IRL of constituent systems. Finally, the proposed method mentioned in section 3.3.3 allows us to estimate the requirement interdependency strengths. For simplicity, we skip over the detailed process entailed to obtain these

values. The table in Figure 23 summarizes initial failure rates for all systems and conditional distributions of failure rates in terms of beta distributions, and expected development time for each system.

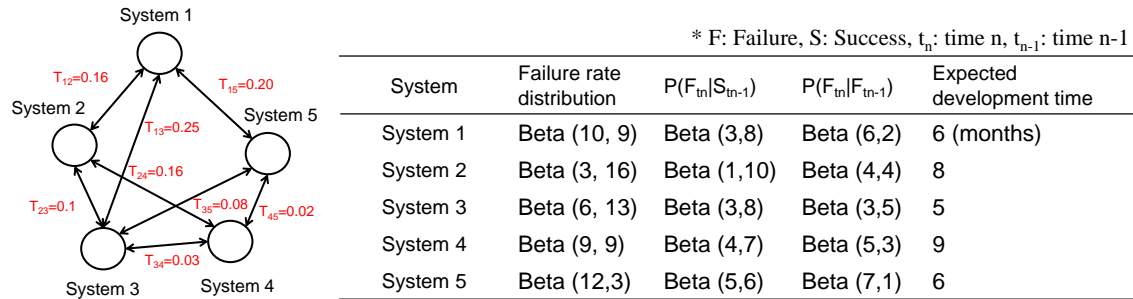


Figure 23. A five system development network and all input information for the analysis

Consider the information fusion of failure rates of system 2, 3, and 5 with system 1 (

Figure 24). System 1 is connected to three dependent systems, 2, 3, and 5, with interdependency strength of 0.16, 0.25, and 0.20 respectively. Systems 2, 3, and 5 have their own inherent failure rates with beta distributions shown in

Figure 24 (a). The propagating failure rate on system 1 is easily calculated through the proposed approach, based on the given information about inherent failure rates and conditional probability for propagating effects. In

Figure 24, blue lines denote the inherent failure rate distributions for systems and red lines denote the propagating failure rate on system 1. The green line in

Figure 24 (b) represents the integrated failure rate for system 1. The mean of the system 1 integrated failure rate represents an increase of 0.26 over its inherent rate value due to the propagating effects from dependent system failures. Figure 25 shows the mean values of propagating effects for all systems. These values depend on the number of dependent systems and the failure rates of dependent systems. System 1 has the highest propagating effects, indicating strong dependencies with numerous other systems. It also has a higher probability to be disrupted by the failure of other systems during the development process. Since it is hard to control this kind of failure, the design team for system 1 should consider these propagating effects when scheduling the development time.

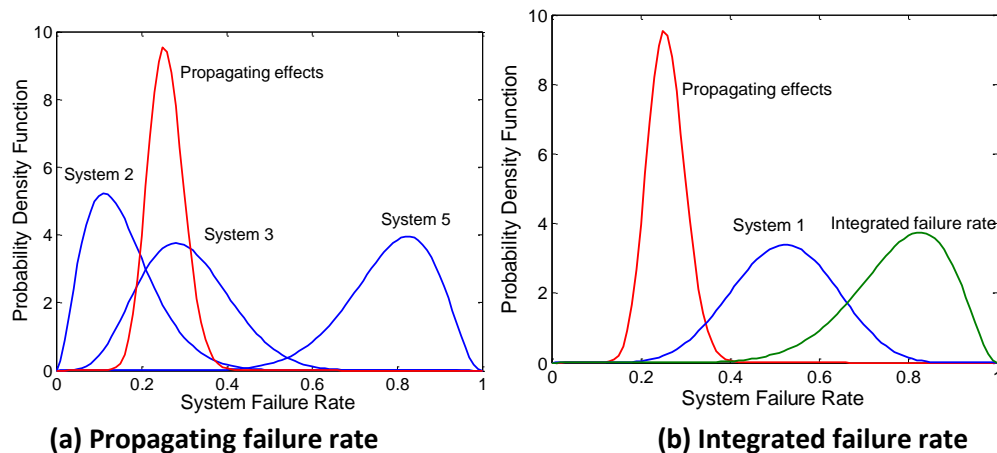


Figure 24. Information fusion of failure rates of system 2, 3, and 5 with system 1

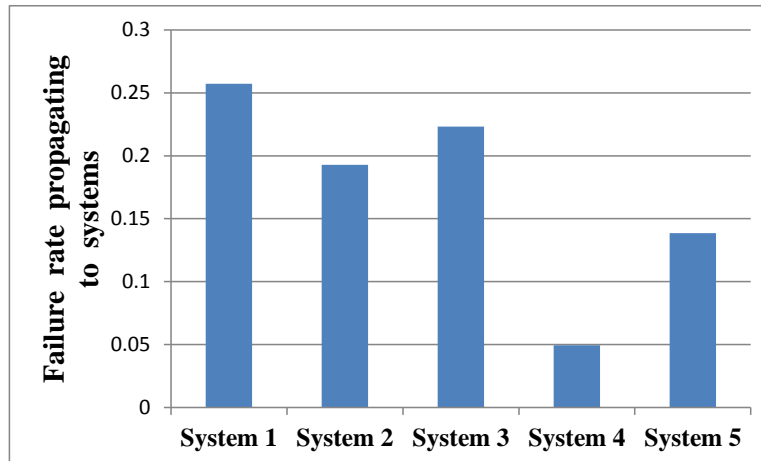


Figure 25. Mean values for failure rates propagating to systems

The same synthetic problem is now used to measure the effects of disruptions and dependencies. The total expected development time is adopted to measure development time of a SoS capability. We assume that each system manager sets up the expected development time for their system as defined in the table in Figure 23, indicating that in the absence of any failures, each system can be completed in the expected development time. Each system also has a development delay time due to its failure rate, calculated as the product of failure rate and the expected development time. For instance, if a system's failure rate is 0.6 and expected development time for the system is 1, then the project team of this system needs 1.6 times the normal duration to complete it. Therefore, the total expected development time can be formulated as the follows:

$$\text{Total expected development time} = \sum_i (1 + \text{failure rate}_i \times \text{expected development time}_i) \quad (20)$$

Table 1 shows the expected development time with/without considering disruption and dependency effects. The disruption and dependency effects increase total development time by 1.6 times. This schedule overrun usually ends up with cost overruns. Therefore when decision makers take a decision on a new SoS capability, disruption/dependency levels of required systems should be considered.

Table 1. Disruption and dependency effects to expected development time

Systems	Expected development time (months)		
	without disruption & dependency effects	with disruption effects & without dependency	with disruption & dependency effects
System 1	6	9.1	10.7
System 2	8	9.2	10.8
System 3	5	6.5	7.6
System 4	9	13.4	13.9
System 5	6	10.7	11.5
Total	34	48.9	54.5



Suppose there are 10 time steps for scheduling checks during the development process. In the beginning of the SoS development, it is hard for decision makers to estimate whether this project will end on time or not due to the high uncertainty. However, as time goes by, decision makers gain confidence in the results. Figure 26 (a) shows this pattern using the mean values of the integrated failure rate of system 4 with 95% confidence interval. In this case, we use the best case scenario to run the simulation, indicating that all schedules at every time step are met on time. These observations are also applied to update failure rate of system 4. Figure 26 (b) represents three different cases - best, worst, and expected, of integrated failure rates of system 4. All observations in the worst case are failures to achieve the schedule at each time step. We also draw numbers from beta distribution of integrated failure rate at the previous step to define the expected case. System 4 can lie anywhere in between best and worst cases. If the integrated failure rates of system 4 shows an increasing pattern then decision makers need to devote more attention to system 4 or substitute it with an alternative system.

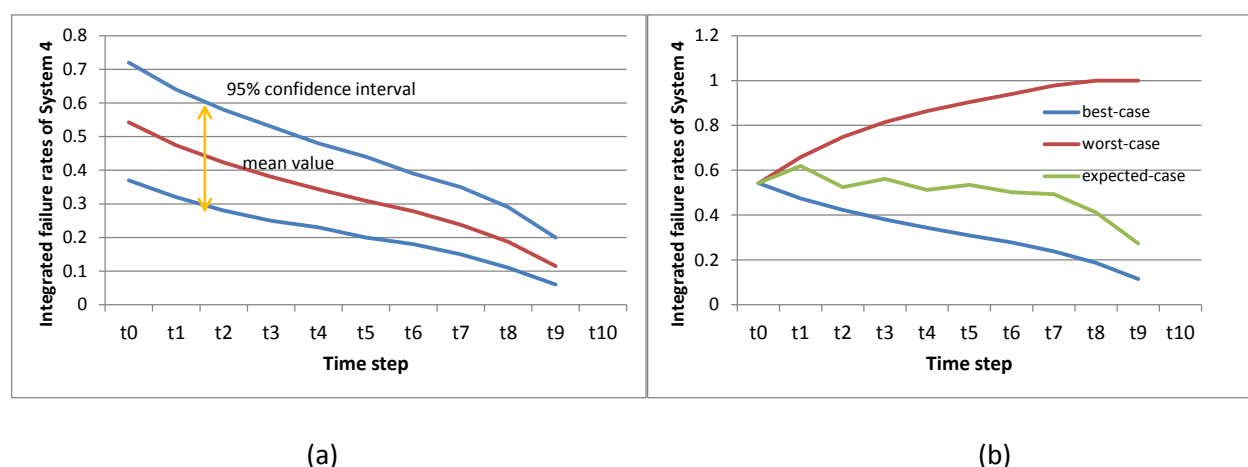


Figure 26. (a) Mean values of integrated failure rate of system 4 with 95% confidence interval, (b) Mean values of integrated failure rate of system 4 in three different cases

### 3.3.7 CONCLUSION AND FUTURE WORK

The development process of a system-of-systems capability is often affected by risks arising from interdependencies between constituent systems during the development life cycle. Research in this section adopts a Bayesian Network approach to analyze interdependencies using constituent system failure rates and requirement dependency strengths between systems. Propagating failure rates are calculated to describe interdependencies, with inherent failure rates being evaluated for individual systems. By the integration of these two failure rates, both currently expressed as beta distributions, a new failure rate distribution is achieved that holistically represents the true risk and accurately determines the critical components. Uncertainty represented in a Bayesian Network using beta distributions allows for enhanced robustness of model outcomes. Event trees are used to show

evolution of constituent system failure rates during the development process. Results that consider both evolution of constituent system failure rates and propagating effects can help to manage risk in development of a SoS with complex interdependencies.

A simple, synthetic five-system SoS problem demonstrates the proposed framework. Results show the increase in integrated failure rate due to the propagating effects of interdependencies. The comparison of integrated failure rates among all systems is useful in identifying the most critical system in terms of which system generates the most vulnerability to propagating effects from dependent systems.

The specific Bayesian Network formulation approach in this research rests on two basic assumptions. First, systems can only have one of two discrete states, such as 'working' or 'failure'; thus continuous variables such as development percentage cannot be expressed directly. Second, the interdependency strengths between systems are assumed constant; thus there is no evolution of the interdependency strengths. Future work will address the relaxation of these assumptions. Furthermore, there is a need to develop a strategic action tool, as shown in the overview of the integrated simulation model in Figure 17. This tool will allow decision makers to take better decisions in a SoS development process.

---

## 3.4 A MARKOV MODEL TO ANALYZE DEVELOPMENT INTERDEPENDENCIES IN SYSTEM-OF-SYSTEMS

Markov-Chain-like Networks have been used to identify interdependency characteristics, evaluate how disruptions propagate in complex networks of interdependent systems, quantify the cascading effects of development risk, and compare networks of systems in their ability to reduce the impact of risk.

---

### 3.4.1 INTRODUCTION

The interdependencies between constituent systems form networks that, while enabling capabilities that are beyond those of individual systems, also increase risk since disruptions (failures, delays, requirement evolution) may propagate to other directly or indirectly dependent systems. Therefore, network analysis tools can help to describe the property of a network, to identify critical or vulnerable constituent systems or aggregations of systems and to individuate convenient metrics to compare different networks.

In this proposed approach, the network of interdependencies is based on a Markov process model. A Markov process is a stochastic process with the Markov property; this property, also called *memorylessness*, means that the conditional probability distribution of future states of the process depends only upon the present state, not on the sequence of events that preceded it. Thus, a Markov chain, i.e. a Markov process with a discrete state-space, is completely defined by the initial probabilities for each state and the matrix of transition probabilities.

The proposed approach characterizes the network as a Markov chain, represented by a directed graph with Markov property. The states, depending on the analyzed level, are the constituent systems (low level) or the requirements (medium level), while the edges are the interdependencies between states, the transition probabilities are the dependency strengths between systems, and the edge weights assess the impact caused by the dependency.

Due to the Markov property, the transition probability between two states is not depending on the previous evolution of the process, but only on the current state; nonetheless, the probabilities may change with time, due for example to change in requirements or in technologies. The Markov chain so built is used in the proposed approach to develop and test metrics suitable both to individuate critical constituent systems, requirement, interdependencies paths (system-level metrics) and to compare different networks (network-level metrics).

---

### 3.4.2 ANALYTIC FRAMEWORK

The metric is developed according to the formulation by (Mane, DeLaurentis, & Frazho, 2011). A typical network is shown in Figure 27. An additional state is added, representing the absorption of the disruption.

A single disruption can be generated in any node of the network, with a given probability, and is then propagated along the network, until absorbed. Required inputs are:

- the matrix A of transition probabilities.  $A_{ki} = P(x_k(n+1)|x_i(n))$ .

- the vector  $b$  of probabilities of initial disruption arising in a given node.  $b_j = P(x_j(0))$ .
- the vector  $c$  of probabilities of disruption to be absorbed by a given node.  $c_i = P(x_F(n + 1/xin))$ .

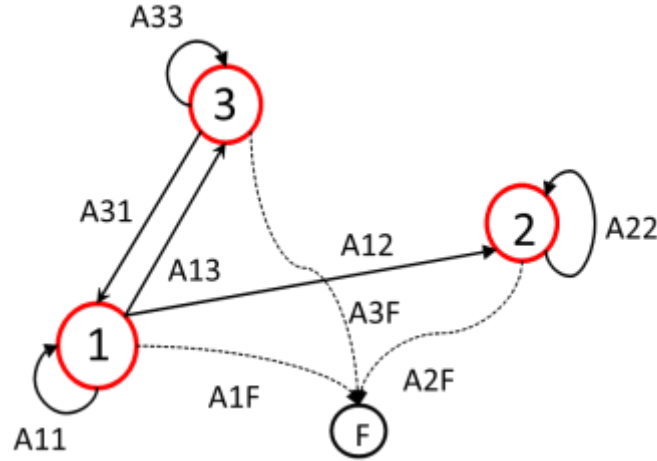


Figure 27. A Markov network. F is the absorbing state, T's are the transition probabilities, D's are the weights (importance of the disruption being propagated through a link)

The expected time before disruption absorption is computed for each node, by Markov property (Mane, DeLaurentis, & Frazho, 2011), as

$$E(F|x_j(0)) = \sum_{n=0}^{\infty} nP(x_F(n)|x_j(0)) = \sum_{n=0}^{\infty} ncA^n b = cA(I - A)^{-2}b \quad (21)$$

The nodes can then be ranked based on their criticality in the propagation of disruption that is higher if the expected time before disruption absorption is higher.

The total expected time before disruption absorption for the entire network is the sum of all expected time (one for each node), each multiplied by the probability of disruption arising in the corresponding node. This formulation is suitable to compare different architectures for development networks.

This Markov model can be applied to any kind of architecture, given that the Markov property is realistic, and it is better suitable for directed/collaborative SoS type. Limitations of the model are the hypothesis that the disruption is propagated to a single state per time, and the low flexibility to addition or deletion of a node, since an entire new transition matrix is required, to satisfy the Markov property.

### 3.4.3 APPLICATION: AN ILLUSTRATIVE EXAMPLE

The method has been applied to a small five-node network, ranking each node based on the expected time before disruption absorption when the initial disruption originates in that given node. The network is organized according to two different architectures (Figure 28): a *whisper* architecture, where the nodes are directly connected to each other (and the disruption directly propagates through the nodes), and a *blackboard* architecture, where the connection between each pair of nodes flows through a common blackboard node.

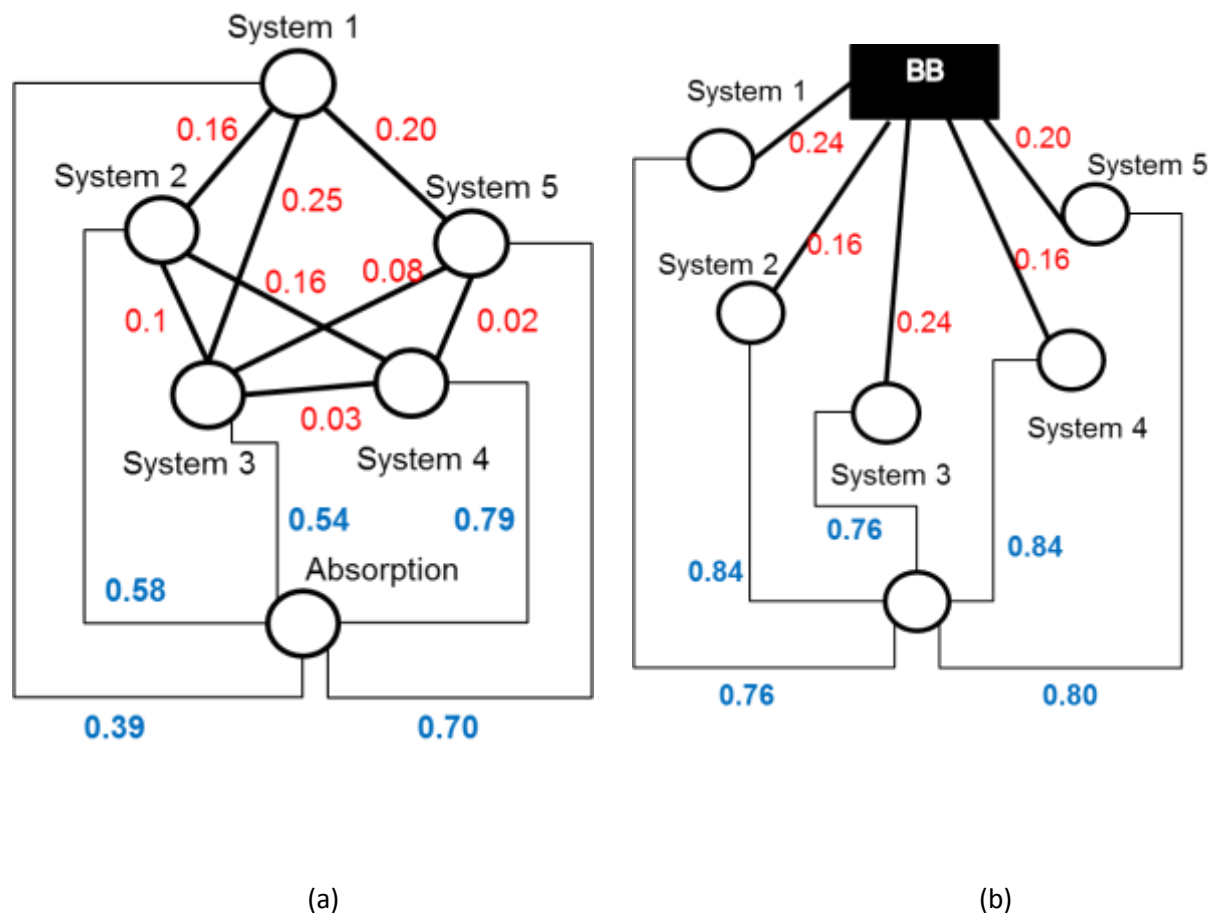


Figure 28. (a) Five-node whisper architecture, (b) Five-node blackboard architecture

The results (Table 2) show that the nodes present different criticality if the architecture changes, with node 1 being the most critical in the whisper architecture, and node 3 being the most critical in the blackboard architecture.

Table 2. Node ranking by Markov analysis in whisper and blackboard architectures

	Whisper		Blackboard	
	Expected time	Impact rank	Expected time	Impact rank
<b>System 1</b>	0.2118	1	1.6799	2
<b>System 2</b>	0.1467	3	1.5183	4
<b>System 3</b>	0.1712	2	2.0405	1
<b>System 4</b>	0.0730	5	1.4261	5
<b>System 5</b>	0.1175	4	1.5970	3

### 3.4.4 CONCLUSION AND FUTURE WORK

A Markov-chain-like network representation for SoS, and a metric to assess the cascading effects of disruptions in a SoS development network have been successfully surveyed and tested.

Results showed that this method can classify nodes based on the expected time before disruption absorption that gives a measure of the impact of the node on the cascading effect of disruptions in the network. In addition, since the metric gives an evaluation of the robustness of the entire architecture, it is suitable to compare different architectures.

After this preliminary survey and application, such method has not been further extended: as described in the previous sections, Bayesian networks have been used to develop probabilistic metrics to analyze interdependencies, while a new method to analyze development dependencies is presented in section 3.6.

### 3.5 DEPENDENCY NETWORK ANALYSIS

A Dependency Network Analysis technique (Garvey & Pinto, 2009) has been adapted to assess operability, reliability, and resilience in both operational and development networks, associated with System of System architectures.

The architecture is modeled as a directed network where nodes represent either the component systems or the capabilities to be acquired. Links on the network represent various kinds of dependencies between the constituent systems: functional dependency in an operational network, sequential development dependency in a development network. Each dependency is characterized by strength and criticality. The ultimate goal of the technique seeks to analyze effects of such dependencies -and of their strength and criticality- on operability, and to identify valid operating and developing strategies and architectures. For operational networks, Functional Dependency Network Analysis (FDNA) is used to assess the effect of topology and of possible degraded functioning of one or more systems on the operability of the network. For development networks, Development Dependency Network Analysis (DDNA) is used to assess how development time or capabilities are affected by the network topology and by delays in the development of component systems.

Each technique is evaluated with regard to amount and quality of necessary input, completeness and usefulness of results, and applicability to problems of diverse nature.

### 3.5.1 FUNCTIONAL DEPENDENCY NETWORK ANALYSIS

#### 3.5.1.1 Analytic Framework

In FDNA, the SoS is represented as an operational network, with the nodes being component systems or capabilities to be achieved. The links represent a dependency of the operability of a node (successor node) from another (predecessor node).



Figure 29. Operational dependency of node  $N_j$  from node  $N_i$

The operability of a node is defined as the “percentage” of effectiveness, that is the level at which the system is currently operating, or the level at which the desired capability is being currently achieved. Operability, ranging between 0 and 100, can be related to performance by means of an input function. In the example in Figure 30, a planetary probe communication system is evaluated based on the number of valid data downlinks per week (performance), with its operability (effectiveness) being 100 when the system performs 1000 valid data downlinks per week, and the degraded operability being 25 when the system performs 380 valid data downlinks per week.

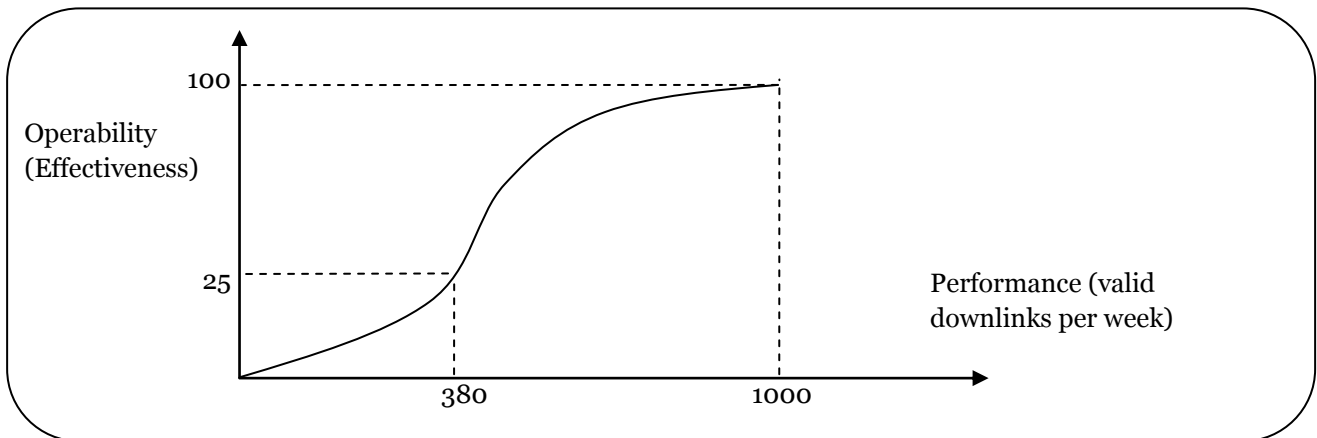


Figure 30. Correlation between Performance and Operability of a system.

Using this relationship, the operability of root nodes, i.e. nodes that have no predecessors, can be evaluated based on their current performance. FDNA analysis, then, allows for the computation of successor nodes operabilities, which can in turn provide a value for the nodal performance. In this

research, the function relating performance and operability is assumed to be a given input, and the method is described on the basis of the operability.

Further input is required:

for each node  $N_i$ , a self-effectiveness level  $SE_i$  is needed, ranging between 0 and 100. For root nodes, this is just the operability; for nodes that have at least one predecessor, the self-effectiveness is the level of operability that the node would have, if it were a root node: therefore, the self-effectiveness assess the current status of a node, not accounting for its dependencies.

For each link, two values are needed. The strength of dependency (SOD) between node  $N_i$  and node  $N_j$ ,  $\alpha_{ij}$ , and the criticality of dependency (COD) between node  $N_i$  and node  $N_j$ ,  $\beta_{ij}$ .  $\alpha_{ij}$  must be between 0 and 1, and can be evaluated as the fraction of the operability level of node  $N_j$  due to the dependency by node  $N_i$ . For example, if the operability level of node  $N_j$  (working at self-effectiveness equal to 100) is 70, then  $\alpha_{ij}$  is equal to 0.3.  $\beta_{ij}$  must be comprised of values between 0 and 100, and it can be evaluated as the maximum level of operability reachable by node  $N_j$ , when the operability of node  $N_i$  is 0. A lower value  $\beta_{ij}$  corresponds to a higher criticality of dependency.

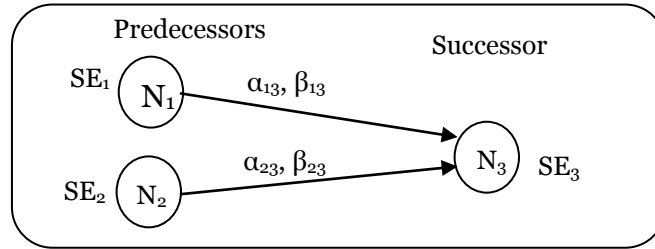


Figure 31. A small network, showing the required input for FDNA

According to the method proposed by Garvey and Pinto, the operability of root nodes is simply their self-effectiveness:

$$O_i = SE_i \quad (22)$$

The operability of nodes that have at least one predecessor is computed as the minimum of two terms, one depending on the SODs, one depending on the CODs:

$$O_j = \min(SOD\_O_j, COD\_O_j) \quad (23)$$

For a node  $N_j$  having  $n$  predecessors, the two terms are computed according to equations (24) - (27):



$$SOD\_O_j = \text{Average} (SOD\_O_{j1}, SOD\_O_{j2}, \dots, SOD\_O_{jn}) \quad (24)$$

$$SOD\_O_{ji} = \alpha_{ij}O_i + (1-\alpha_{ij})SE_j \quad (25)$$

$$COD\_O_j = \text{Min} (COD\_O_{j1}, COD\_O_{j2}, \dots, COD\_O_{jn}) \quad (26)$$

$$COD\_O_{ji} = O_i + \beta_{ij} \quad (27)$$

The term accounting for SOD is the average operability values of node  $N_j$ , computed for each dependency from a predecessor node  $N_i$ , thus reflecting the relationships between the  $n$  predecessors and the node  $N_j$ . The term accounting for COD is the minimum of the values of the operability of node  $N_j$  computed for each dependency from a predecessor node  $N_i$ , thus precisely reflecting the importance of the most critical dependency.

Using equations (22) - (27), the operability of each node can be sequentially computed, starting from the root nodes, in a breadth-first way: after the roots, nodes directly depending from the root are analyzed, and so on.

### 3.5.1.2 Application

To assess the value and applicability of FDNA for SoS, the method is applied to a simple five-node aerospace network (Figure 32).

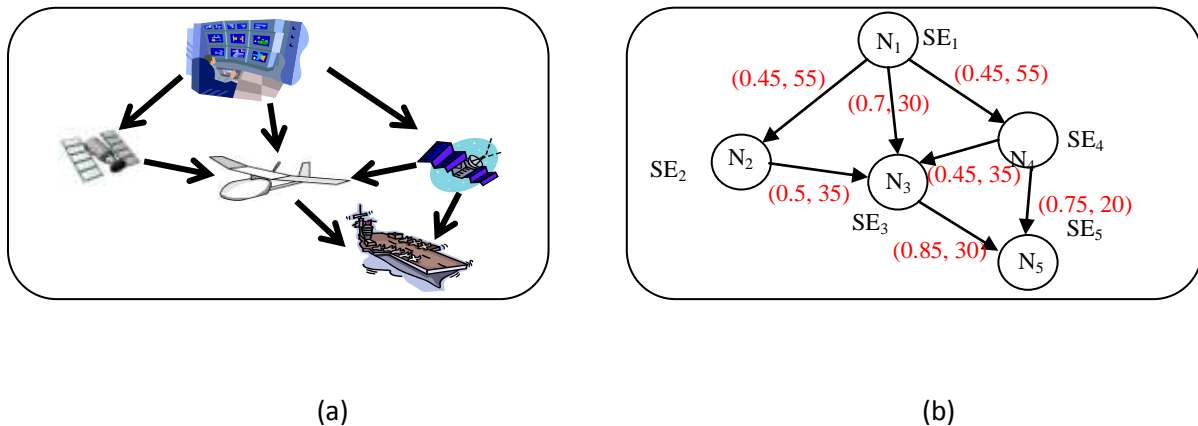


Figure 32. (a) the five-node aerospace SoS; (b) the same SoS represented as a network, with the required input for FDNA: self-effectiveness of each node, and COD and SOD of each link.

The network includes ground facilities ( $N_1$ ), two satellites ( $N_2$  and  $N_4$ ), a UAV ( $N_3$ ), and a ship ( $N_5$ ). The links represent communication and data dependencies for location: the satellites and the UAV need data from the ground facilities, the UAV also uses the satellites for navigation, and the ship gets data

from the UAV and from one of the satellites. The input can be given in form of a vector for the self-effectiveness, and two matrices containing the terms  $\alpha_{ij}$  and  $\beta_{ij}$  (nonzero entries in the SOD matrix correspond to dependencies in the network).

$$SE = [SE_1 \ SE_2 \ SE_3 \ SE_4 \ SE_5] \quad SOD = \begin{bmatrix} 0 & 0.45 & 0.7 & 0.45 & 0 \\ 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.85 \\ 0 & 0 & 0.45 & 0 & 0.75 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad COD = \begin{bmatrix} 0 & 55 & 30 & 55 & 0 \\ 0 & 0 & 35 & 0 & 0 \\ 0 & 0 & 0 & 0 & 30 \\ 0 & 0 & 35 & 0 & 20 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

## Deterministic analysis

In this kind of simulation, different values for the self-effectiveness, i.e. degraded operability of system, are fed into the equations, to compute the actual operability of each system. Table 3 shows the results of degraded operability (level of operability equal to 75 or to 25) in a single system.

Analysis of these results give good insight into the influence of dependencies into such network: first of all, the five-node SoS appears to have high resilience in these cases: any single failure that degrades the operability of a system down to 25, can at most degrade the operability of the ship down to 68.22. The second result from FDNA analysis is the identification of the most critical nodes, in this case nodes  $N_1$  and  $N_4$  most affect the operability of other nodes. While some result is anticipated (for example, since there is a path from node  $N_1$  to any other node, its influence is expected), other behaviors are captured by FDNA. For instance, node  $N_4$  shows positive criticality for what concerns the dependency of node  $N_5$ : when the satellite  $N_4$  is working at level 100, the operability of the ship is never lower than 85, even if other systems influencing  $N_5$  are degraded. Finally, FDNA allows more complex analysis, based on the output of interest: for example, it can be noted that, if the operability of the ship is the object of the evaluation, highly degraded operability of node  $N_3$  is worse than highly degraded operability of node  $N_2$ ; if instead the global operability of the network, computed as the sum of the operability of each node, is the measure of interest, highly degraded operability of node  $N_3$  gives better results than highly degraded operability of node  $N_2$  (noticeably, this is not true if the operability decreases to 75. This is due both to the CODs and SODs, and to the topology, sources of complexity in the network. To better catch these details, stochastic analysis can be performed).

Table 3. Degraded self-effectiveness in single systems. O=operability of the nodes

Self-effectiveness [ $N_1 \dots N_5$ ]	$O_1$	$O_2$	$O_3$	$O_4$	$O_5$
[75 100 100 100 100]	75	88.75	90.60	88.75	91.79
[25 100 100 100 100]	25	66.25	55	66.25	68.22
[100 75 100 100 100]	100	86.25	97.71	100	99.03
[100 25 100 100 100]	100	58.75	93.13	100	97.08
[100 100 75 100 100]	100	100	87.5	100	95.22

[100 100 25 100 100]	100	100	66.25	100	85.66
[100 100 100 75 100]	100	100	97.94	86.25	93.97
[100 100 100 25 100]	100	100	93.75	58.75	78.75
[100 100 100 100 75]	100	100	100	100	95
[100 100 100 100 25]	100	100	100	100	85

Table 4. Examples of degraded self-effectiveness in pairs of systems. O=operability.

Self-effectiveness [N <sub>1</sub> ... N <sub>5</sub> ]	O <sub>1</sub>	O <sub>2</sub>	O <sub>3</sub>	O <sub>4</sub>	O <sub>5</sub>
[25 25 100 100 100]	25	25	55	66.25	68.22
[25 100 25 100 100]	25	66.25	38.06	66.25	61.02
[25 100 100 25 100]	25	66.25	55	25	45
[25 100 100 100 25]	25	66.25	55	66.25	53.22
[100 25 25 100 100]	100	58.75	59.38	100	82.73
[100 25 100 25 100]	100	58.75	86.94	58.75	78.75
[100 25 100 100 25]	100	58.75	93.13	100	82.08
[100 100 25 25 100]	100	100	60.06	58.75	67.56
[100 100 25 100 25]	100	100	66.25	100	70.66
[100 100 100 25 25]	100	100	93.75	58.75	66.88

In Table 4, results achieved with deterministic analysis of the network with two degraded systems are shown. Once again, even this small experiment shows useful results: besides confirming nodes N<sub>1</sub> and N<sub>4</sub> as the most critical, the analysis shows that a failure in node N<sub>5</sub> is not anymore well absorbed if there is a concurrent degraded functioning in node N<sub>1</sub>. Globally, however, the SoS shows a good capability of reacting to degraded operability.

## Stochastic analysis

A more realistic view of the behavior of a SoS as a function of the dependencies between the component systems can be achieved by means of a stochastic analysis with the FDNA technique. This kind of analysis is useful to capture and summarize all the aspects related to SOD and COD, as well as to topology, in a few probability distributions (as reported above, such details are spread in the tables for deterministic FDNA). Since the five-node network is very small, a Monte Carlo simulation has been executed.

The computation of the expected value for the operability of a system gives a measure of the resilience of such system to failures of the predecessors, while the variance evaluates the sensitivity of the system to failures of the predecessors. Differently from the deterministic analysis, this evaluation is not based on the simulation of single instances (that could for example neglect some COD), but it accounts for any possible combination of the effects of COD, SOD, and topology. The capability of this kind of analysis to

capture behavioral patterns and features of a whole architecture makes it suitable to be used as a decision tool.

The analysis confirmed that the systems in the five-node network are resilient to single failures in the predecessors, with a probability distribution shifted towards high values of operability (Figure 33 (a)).

The complexity of the interdependencies between systems, resulting in more complex behavior, arises even if only two system experience a degraded operability in the simple five-node network, as shown in Figure 33 (b).

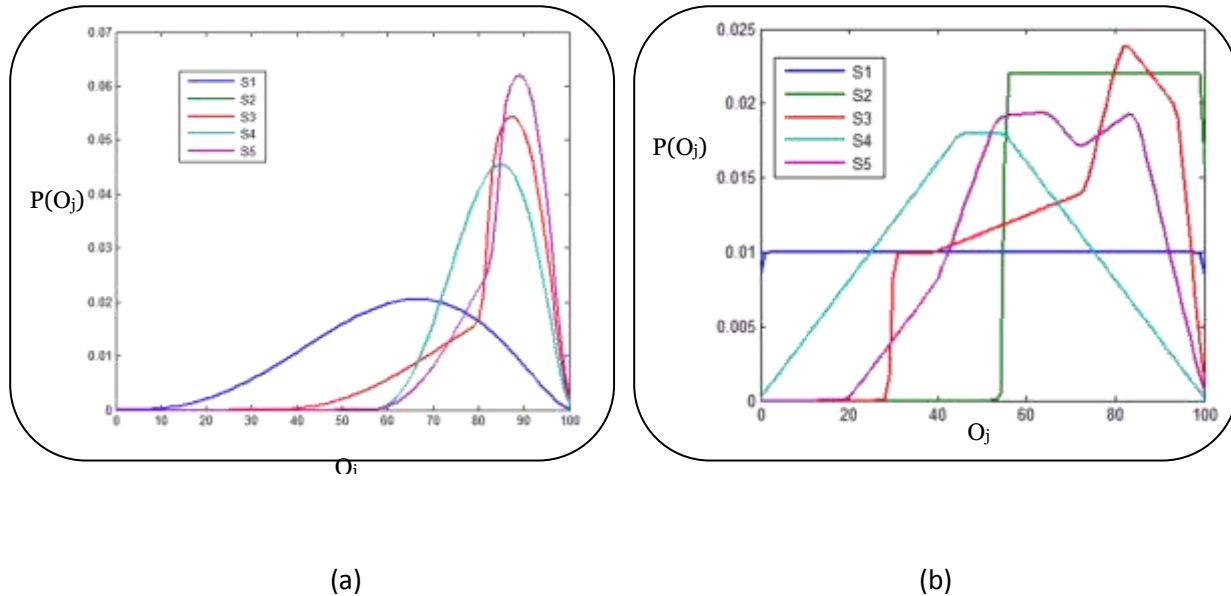


Figure 33. Probability distribution for the operability in the five-node system. (a) single failure in node  $N_1$ , beta probability distribution for self-effectiveness of node  $N_1$ ; the probability distribution of S2 coincides with that of S4; (b) failure in nodes  $N_1$  and  $N_4$ , with independent uniform distribution: node  $N_3$  is in this case more resilient than node  $N_5$ , differently from what appears in 32a

## Example of greater complexity

An additional scenario is executed using a more complex twelve-node network (Figure 34), with three ground facilities ( $N_1, N_2, N_3$ ), two satellites ( $N_4, N_5$ ), two UAVs ( $N_6, N_7$ ), a ship ( $N_8$ ), an airplane ( $N_9$ ), and three nodes corresponding to desired capabilities: long range detection ( $N_{10}$ ), short range detection ( $N_{12}$ ), and rescue ( $N_{11}$ ). The results further demonstrate the power of FDNA in comparing different architectures, as well as the result of removal of nodes or dependencies from a system. Four deterministic tests are reported and results are shown in Table 5.

Interesting outcomes have been achieved, showing unexpected behavior and patterns not directly predictable through the knowledge of the component systems, that is one of the definitions of emergence in the SoS. If node  $N_7$ , i.e. one of the UAVs, is removed from the SoS (meaning that the short range observation is based only on the ship and the airplane), when node  $N_1$  has degraded self-

effectiveness, the operability of node  $N_{11}$  unexpectedly increases. The operability of node  $N_{12}$ , instead, slightly decreases when nodes  $N_1$ ,  $N_5$ , or  $N_8$  are self-effective at level 20, but node  $N_{12}$  is not anymore affected by degraded self-effectiveness of node  $N_2$ . Therefore, such architecture could be preferable if nodes  $N_1$  and  $N_2$  is prone to failures.

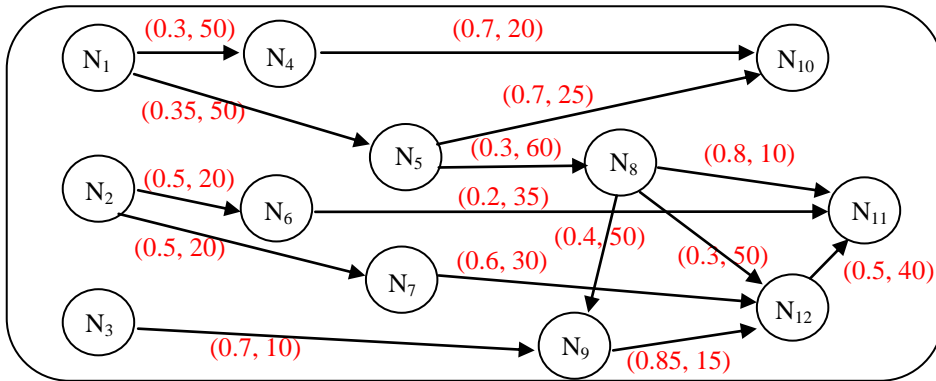


Figure 34. The twelve-nodes SoS

When the dependency of node  $N_9$  on node  $N_8$  is removed, the operability of nodes  $N_{11}$  and  $N_{12}$  globally increases. Noticeably, if node  $N_9$  is having a degraded self-effectiveness, its dependency from a single node ( $N_3$ ) gives a high operability, resulting in better operability of nodes  $N_{11}$  and  $N_{12}$ . Finally, a small architecture change has been tested: node  $N_5$  is dependent from node  $N_2$  instead than  $N_1$ , and node  $N_6$  is dependent from node  $N_1$  instead than  $N_2$ . This new architecture gives lower operability in some case, but it can be noticed that node  $N_{10}$  has a smoother behavior, with the operability being the same for degraded self-effectiveness of nodes  $N_1$  and  $N_2$ . The architecture can then be evaluated based on the importance of the requested capabilities ( $N_{10}$ ,  $N_{11}$ ,  $N_{12}$ ), and on the probability of failures of  $N_1$  and  $N_2$ .

Table 5. Twelve-node SoS experiment: indicated system is having a self-effectiveness of 20 (single failure in other systems gave the same results for all the experiments), while all other systems have self-effectiveness equal to 100

Test	$O_{10}$ , $O_{11}$ , $O_{12}$ ( $N_1=20$ )	$O_{10}$ , $O_{11}$ , $O_{12}$ ( $N_2=20$ )	$O_{10}$ , $O_{11}$ , $O_{12}$ ( $N_5=20$ )	$O_{10}$ , $O_{11}$ , $O_{12}$ ( $N_8=20$ )	$O_{10}$ , $O_{11}$ , $O_{12}$ ( $N_9=20$ )
Basic network	79, 96.36, 98.59	100, 75, 70	69, 93.2, 97.37	100, 54, 91.23	100, 96.5, 79
Node removal	79, 97.25, 97.89	100, 75, 100	69, 93.2, 96.05	100, 54, 86.84	100, 96.5, 79

Link removal	79, 99.1	97.45, 100, 75, 70	69, 98.32	93.2,	100, 54, 94	100, 98.5, 91
Diff. architecture	89.5, 75, 100	89.5, 92.6, 70	69, 97.37	93.2,	100, 54, 91.23	100, 96.5, 79

### 3.5.2 DEVELOPMENT DEPENDENCY NETWORK ANALYSIS

A Development Dependency Network Analysis (DDNA) method, based on the concepts of SOD and COD from FDNA, is developed. It is applied to development SoS networks, where the links, like in PERT, represent development dependencies between systems. The outcome of such analysis is the beginning time and the completion time of the development of each system, as well as an assessment of the combined effect of multiple dependencies and possible delays in the development of predecessors. As in FDNA, this method evaluates the most critical nodes and dependencies, and can be used to compare different architectures in term of development time. Also, if some of the nodes are capabilities to be achieved, the method assess the time, or the expected time, in probabilistic analysis, by which each capability is available. The method has been developed to also account for the possibility of measuring partial capabilities attained during the development of a SoS.

#### 3.5.2.1 Analytic Framework

Compared to existing methods, such as PERT/CPM, DDNA provides more specific insight into the effects of multiple and diverse dependencies on the development of SoS. These include:

The strength of dependency affects both the beginning time and the completion time of development of a system. Differently from PERT, development of a system can start before a predecessor is complete, if the dependency has not reached criticality. Therefore, a more realistic analysis of development time is achieved. Above all, this feature allows DDNA to assess partial capabilities during the development of a SoS.

COD affects the beginning time in the same way as in PERT/CPM: a successor must wait until a critical predecessor is complete to begin development. Instead SOD results in a less absolute dependency.

Differently from FDNA (where performance is related to operability), in DDNA time is directly used into the computation, and the level of performance constitutes an assessment of the quality of the development (for example, three weeks could be the best time for a system, ten weeks could be the worst, but eight weeks could correspond to a satisfaction of 50%).

In DDNA, each node requires three pieces of input data: the minimum independent time (MINIT), that is the minimum duration of development of the system; the maximum independent time (MAXIT), that is the maximum duration of development of the systems; the self-effectiveness (SE) that linearly evaluates how much the system is close to being developed with its minimum duration time, without accounting for dependencies. If SE=0, then the independent duration of development is equal to MAXIT. If SE=100, then the independent duration of development is equal to MINIT. Due to dependencies, the actual time

to develop a system can be longer than MAXIT (but in this case the system must have begun its development before the completion of a predecessor, thus resulting in earlier partial capabilities), whereas it can never be shorter than MINIT.

Each link requires two input data: the strength of dependency (SOD) and the criticality of dependency (COD). The SOD evaluates how much a system can begin its development before the completion of the predecessor, it constitutes a parameter for the parabolas shown in Figure 35 (a). This shape has been chosen because it results in anticipated development for systems depending from a node with medium-high SE, while no early development is allowed when the predecessor is being developed either in its best time, or with a SE below the critical threshold. In future research, these curves will be compared to data from industrial projects and consequently adapted.

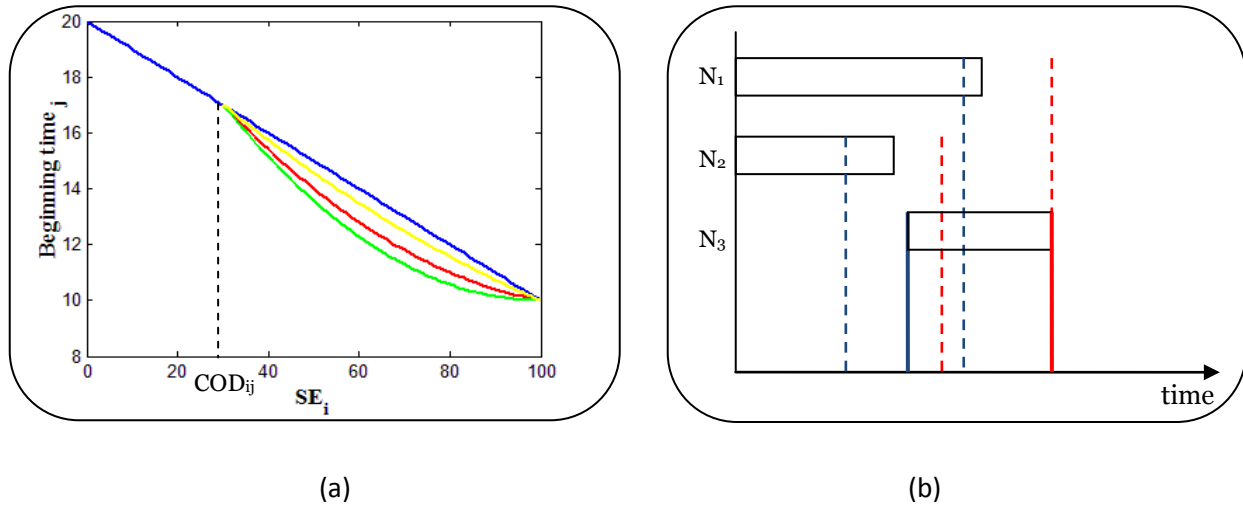


Figure 35. (a) the dependency between node  $N_i$  and node  $N_j$ .

If  $SE_i$  is lower than  $COD_{ij}$ , then the beginning time of  $N_j$  coincides with the completion time of  $N_i$ . Otherwise, the development of  $N_j$  can begin earlier: the blue line relates  $SE_i$  to the completion time of  $N_i$  (coincident with the beginning time of  $N_j$  when  $SOD_{ij}=1$ ). The yellow, red, and green parabolas correspond respectively to  $SOD_{ij}=0.7$ ,  $SOD_{ij}=0.3$ ,  $SOD_{ij}=0$ ; (b) Multiple dependency of  $N_3$  from  $N_1$  and  $N_2$  in a Gantt diagram fashion. Blue lines are the beginning times, red lines are the completion times (dotted=due to single dependency, continuous=actual times due to multiple dependency)

The COD is the minimum level of SE of the predecessor that allows an early development of the successor. If the SE of the predecessor is lower than the COD, then the successor must wait until the predecessor is fully developed. Higher COD corresponds to higher criticality.

For root nodes  $N_i$ , the beginning time ( $BT_i$ ) is 0, and the actual completion time ( $CT_i$ ) is computed as

$$CT_i = MINIT_i + (100 - SE_i) (MAXIT_i - MINIT_i) / 100 \quad (28)$$

That is, depending on its SE. For nodes having a single predecessor, the beginning time is computed according to the function shown in Figure 35 (a): if  $SE_i < COD_{ij}$ , then  $BT_j = CT_i$ . Otherwise,  $BT_j$  is computed as

$$BT_j = a SE_i^2 + (-a (100 + COD_{ij}) + (B - D) / (100 - COD_{ij})) SE_i + (100 - B - COD_{ij}) / (100 - COD_{ij}) + 100a COD_{ij} \quad (29)$$

Where B is the minimum actual completion time for node  $N_i$ , D is the completion time of node  $N_i$  corresponding to  $COD_{ij}$ , and  $a$  is equal to  $(1 - SOD_{ij}) (D - B) / (100 - COD_{ij})^2$ . This formulation corresponds to the parabolas in Figure 35 (a), and guarantees that node  $N_j$  cannot have a beginning time lower than the minimum completion time of node  $N_i$ .

Given a development time of node  $N_j$  equal to

$$DT_j = MINIT_j + (100 - SE_j) (MAXIT_j - MINIT_j) / 100 \quad (30)$$

that is a linear relationship between the SE of a node and its development time, the completion time is

$$CT_j = \text{Max}(BT_j + DT_j, CT_i + SOD_{ij} DT_j) \quad (31)$$

The first term is the completion time that node  $N_j$  would have starting at  $BT_j$  and not having any dependency from  $N_i$ . However, the development time, added to the anticipated beginning time, could result in the successor node  $N_j$  being fully developed before the completion of the predecessor node  $N_i$ . Therefore, the second term accounts for this dependency, stating that node  $N_j$  cannot complete its development before a certain amount of time after the completion of node  $N_i$  elapses (this amount of time being dependent on  $SOD_{ij}$  and development time of  $N_j$ ). Equations (30) and (31) are also used to compute the minimum and the maximum completion time for node  $N_j$ .

For nodes having multiple predecessors, the beginning time is computed as the average of the beginning times given by each dependency. Even if a node has got a critical dependency from one or more predecessors, it can still begin its development based on the dependency from other nodes. Criticality, however, affects the completion time. Using the average, instead than the minimum, prevents a single predecessor from critically influence the beginning time. The completion time is the maximum of the completion times given by each dependency.

### 3.5.2.2 Application

The method has been tested with simple networks to get an insight into the result attainable through this analysis. A deterministic analysis of a five-node network is reported in Table 6 (matrices for the network are stated below).

$$[MINIT, MAXIT] = \begin{bmatrix} 7 & 12 \\ 10 & 20 \\ 6 & 18 \\ 5 & 15 \\ 8 & 20 \end{bmatrix} \quad SOD = \begin{bmatrix} 0 & 0 & 0.7 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0.9 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad COD = \begin{bmatrix} 0 & 0 & 40 & 0 & 0 \\ 0 & 0 & 20 & 0 & 0 \\ 0 & 0 & 0 & 0 & 50 \\ 0 & 0 & 0 & 0 & 30 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$



Table 6. Preliminary results with DDNA method. BT=Beginning Time. CT=Completion Time

Self-effectiveness	BT <sub>1</sub>	CT <sub>1</sub>	BT <sub>2</sub>	CT <sub>2</sub>	BT <sub>3</sub>	CT <sub>3</sub>	BT <sub>4</sub>	CT <sub>4</sub>	BT <sub>5</sub>	CT <sub>5</sub>
[100 100 100 100 100]	0	7	0	10	8.5	16	0	5	10.5	24
[20 100 100 100 100]	0	11	0	10	10.7	17.4	0	5	11.2	25.4
[100 20 100 100 100]	0	7	0	18	12.5	24	0	5	14.5	32
[100 100 20 100 100]	0	7	0	10	8.5	25.6	0	5	16.74	36.48
[100 100 100 20 100]	0	7	0	10	8.5	16	0	13	14.56	24
[100 100 100 100 20]	0	7	0	10	8.5	16	0	5	10.5	33.6
[10 10 100 100 100]	0	11.5	0	19	15.25	25	0	5	15	33
[90 10 100 10 100]	0	7.5	0	19	13.19	25	0	14	19.5	33

Various observations can be done even with this few results: for example, the high SOD between nodes N<sub>4</sub> and N<sub>5</sub>, and N<sub>3</sub> and N<sub>5</sub> causes N<sub>5</sub> to have a longer development than N<sub>3</sub>, even when N<sub>3</sub> starts to be developed later. Also, the capability of the SoS to partially recover from delays is evident, above all when delays strike the root nodes, and successor nodes have low SOD (so they can begin their development, adding partial capability to the overall SoS). In PERT/CPM analysis, a delay on a critical path would never be recovered, nor the analysis would account for the possibility that the System-of-System could be partially developed in the meantime.

### 3.5.3 CONCLUSIONS AND FUTURE WORK

In this section, two methods for the analysis of the effect of dependencies between the components of a SoS have been presented. Functional Dependency Network Analysis is applicable to operational networks, and Development Dependency Network Analysis is applicable to development networks. FDNA quantifies the effect of the dependencies on the operability, and DDNA quantifies the effect of the dependencies on the development time. Both techniques can identify the most critical nodes and dependencies, and their effect on the operability or development of successor nodes, and can compare different architectures. The analysis can be deterministic or probabilistic, and be used to analyze specific or global operability and development behavior. The relationship between performance and operability (or performance and time), and the use of a graph description allows these methods to deal with

problems of different nature and various fields of application: for example, complex aerospace systems, design schedule, industrial production, social relationships can be analyzed by FDNA and DDNA. However, even if both methods are generally applicable to any SoS that can be represented by an operational or development network, virtual SoS (meaning that each component systems is completely independent and is not even partially designed according to the goals of the entire SoS), especially if asynchronous, are hard to treat with such methods, since the SOD and COD of the dependencies would be difficult to evaluate, and could change over time.

Those methods are required to account for typical traits of SoS: complexity, size, partial autonomy in development and operability of the component systems, emergent behavior caused by the complex relationships between the systems.

Output of FDNA and DDNA gives a detailed insight into the effects of dependencies, underlining unintuitive emergent behavior of the SoS, as described in the examples. FDNA can quantify the resilience of a SoS, and evaluate different architectures with respect to their operability when degraded functionality arises. DDNA uses the properties of dependencies (strength and criticality) to assess the development time, and the effect of delays on the development, more realistically than PERT/CPM. When criticality is not reached, partial development of a system can begin before a predecessor is complete. DDNA is able to capture the capability of a network to absorb a delay even along the critical path and is also suitable to assess partial capabilities available during the development of a SoS. However, both methods require many input data: strength and criticality of each dependency in the networks has to be evaluated (by experts, or getting data from simulations, or just using reasonable values), as well as the possible shape of the network. For DDNA, the minimum and maximum development time for each system are further required inputs.

Potential future work will address the following:

- FDNA analysis has to be tested with larger networks, featuring complex interdependencies between component systems; the results can be compared to those of other analysis techniques and metrics, looking for patterns and characteristics relating the features of nodes, links, and architecture of networks (like degree, centrality, weights) to the outcomes of FDNA.
- DDNA analysis is very promising: different shapes for the functions relating the development time of dependent systems can be tested. Also, DDNA will be directly compared to PERT/CPM analysis. Stochastic analysis, though very complex, can be added to this method, in the same way as it has been executed in FDNA. However, the two most important improvements that will be added to this technique involve a tool for optimization vs. cost analysis, similar to that of CPM, and a metric to assess partial capabilities achieved by the SoS during the development of its components.
- For both methods, analysis of data from SoS design found in literature, and comparison with the output from the methods will be performed. In the meanwhile, an Agent Based Model has been developed to represent a naval warfare SoS, and is currently being used to obtain the required input for FDNA.

---

## 3.6 ARCHITECTURE EVOLUTION STRATEGIES ANALYSIS USING COLORED PETRI NETS

The method described in this section employs Colored Petri Nets (CPN), a powerful discrete event dynamic simulation tool, to model, simulate and evaluate the existing and evolving architectures. However, it is unrealistic to improve performance without considering the cost of architecture evolutions. Research in this section regards complexity as an indicator of the architecture evolving cost. Compared to other complexity metrics, dynamic complexity is included in the complexity metric in this section. Eventually, an appropriate evolution choice could be achieved by examining the tradeoff space between complexity and performance. This approach is illustrated with a conceptual SoS problem.

---

### 3.6.1 INTRODUCTION

An architecture is defined as the structure of components, their relationships, and the principles and guidelines governing their design evolution over time (Defense, May 2009). The architecture of an System of systems (SoS) aims to provide a shared persistent representation of the technical framework that guides SoS evolution (Dahmann, Rebovich, & Lowry, May 2011). However, due to the changes in capability and performance objectives, interests of different stakeholders, technology improvements, and unanticipated situations, existing architectures may need to evolve to meet new capability requirements and constraints. The development and evolution of SoS architecture has been regarded as one of the core elements in the SoS system engineering (SE) guide (Dahmann, Rebovich, & Lowry, May 2011) (OUSD(AT&L), August 2008), but remains a challenge because of the complex interactions that exist between SoS constituent entities. Thus, adequate tools are needed in capturing these interdependencies and supporting informed decisions on architectural evolutions.

Diverse sources of changes drive the evolution of SoS architectures through different dimensions. Mangino (Mangino, Bolczak, & Simons, March 2008) identified two dimensions to evolve; organizational and technical. The organizational dimension mainly includes identification of roles for those different stakeholders, while the technical dimension describes the paths of systems/functions, data access, and information technology (IT) infrastructures. Research in this section focuses on technical dimensions, especially those from a system perspective such as adding new systems or functions.

Research work in this section attempts to answer these questions through a simulation framework that allows for these tradeoffs and architectural changes to be quantified. Colored Petri Nets (CPN), a powerful discrete event dynamic simulation tool, is used to model, simulate, and evaluate existing and evolving architectures. CPN has been extensively employed by Levis (Wagenhals & Levis, 2008) to model and evaluate military architectures because it can express executable architectures that correspond to static equivalents represented by static diagrams such as unified modelling language (UML). Research in this section builds on that work to include architectural complexity analysis, a key driver of cost in architecture evolution. CPNs can be used to address evolving architectural complexities and

performance; exploration of the tradeoff space that balances complexity and performance allows for informed decision-making in the evolution of SoS architectures.

---

### 3.6.2 BACKGROUND

Architecture modeling furnishes abstractions to manage complexities, allowing engineers to visualize a proposed system, analyze the problem domain, and describe and specify the architecture for the solution domain (Wang & Dagli, 2011). The Department of Defense (DoD) has developed an Architecture Framework (DoDAF) (Defense, May 2009) to provide support and guideline for architecture development, and has gained great attentions and applications (NextGen Air Transportation System Enterprise Architecture, July 2007). In order to further simplify the process of verification, validation and evaluation of the architectures, executable architecting was developed. Wagenhals and Levis (Wagenhals & Levis, 2008) proposed an executable CPN model to simulate and evaluate a service oriented architecture. Wang and Dagli (Wang & Dagli, 2011) integrated system modeling language (SysML) and CPN into model-driven systems development to create a structured architecture design process. Griending and Mavris (Griending & Mavris, 2011) summarized four common approaches for SoS executable architecting, which are Markov chains, Petri nets, system dynamics models and mathematical graphs. Agent Based Modeling(ABM) is another popular choice for dynamic architecture modeling. DeStefano (DeStefano, 2004) utilized ABM to create an executable model of a weapon architecture. Each of these methods possesses its own advantages and disadvantages. ABM is suitable for representing an environment composed of interactive parties, but it suffers from computational workload. CPN's high level approach to describing models negates the need for large amounts of detailed computations and provides values for understanding the dynamic behavior of a system. In this section, we choose CPN to model the functional architectures for purposes of 1) matching the static architecture to executable architecture in an easy way 2) keeping the architecture modeling in an abstract and high level 3) expressing and observing the dynamic and concurrent operations.

The Federal Aviation Administration (FAA) has large efforts to study architectural evolution of the NextGen Air Transportation System (Mangino, Bolczak, & Simons, March 2008). Organizational and technical dimensions of System Wide Information Management (SWIM) architecture evolution have been presented and divided into five sub-dimensions. Organizational dimension includes SWIM environment and Communities of Interest (COIs). Technical dimension consists of net-enabled services and data, service and data access, as well as IT infrastructure and core services. Likewise, EuroControl developed evolution plan of logical architecture for Air Transportation Management (ATM) driven by the need for increased system performance (European Organization for the Safety of Air Navigation, 2007). It mentioned a couple of ways to achieve the evolving performance such as increasing resources, adopting new concepts of operation, incorporating new technologies, etc. Compared to these explorations by organizations, others focus more on one dimension of architecture evolution. For example, Jain (Jain, June 2011) proposed a BPMN model-based structure ArchEE to make informed evolution decisions of mission oriented IT architectures. Research in this section instead aims to facilitate evolution decision making for functional architectures.

Complexity is an increasingly important aspect of SoS development, and has resulted in much research. Kinnunen (Kinnunen, Feb.2006) gave a great summary of existing complexity metrics and brought up an interface complexity multiplier (ICM) for architecture complexity measurements, including distance, volume of interchange, quality requirements, reliability and such kind of properties to stress the criticality of interface in an SoS. Domercant (Domercant & Mavris, 2010) combined three different complexity metrics together to demonstrate three aspects of SoS architecture, that being: system and functional, interface and structure aspects respectively. However, these complexity metrics primarily focus on the static structure and information. Petri Nets offer a framework to address the complexity of system operations and interactions. Arteta (Arteta & Giachetti, May 2004) addressed business process complexity based on state space probability using Petri net. Ammar (Ammar, Nikzadeh, & Dugan, June 2001) quantified an overall complexity of a software system encompassing both static and dynamic aspects using the dynamic property of CPN. Fry (Fry & DeLaurentis, June 2011) extended Ammar's work to an SoS context. Static complexity captures the degree of transitions while dynamic complexity computes the complexity of the operating functionalities and concurrent activities through obtaining information from the execution of CPN model. Thus not only the static structure of an SoS can be expressed, the dynamic interactions and operations of systems could also be illustrated. From this point of view, research in this section employs this complexity metric and makes some extension.

---

### **3.6.3 PROPOSED APPROACH**

This section describes a simulation framework that addresses SoS performance and complexity. Neither architecture modelling techniques nor complexity metrics are new. However, the unique contribution is the integration of performance and complexity in the same framework to guide the architecture evolution.

#### **3.6.3.1 Colored Petri Nets Modeling**

Petri nets can cope with dynamic processes and concurrent events, and express the information flow of systems through use of circle places, rectangle transitions, directional arcs and dot tokens. Colored Petri net extends Petri net by distinguishing tokens through definitions of different data types called color sets, which provides the possibility of creating compact models (Jensen, Kristensen, & Wells, 2007). CPN are usually accompanied by temporal, hierarchical and stochastic features.

The architecture modeling process can be separated into three steps -- analysis phase, synthesis phase and evaluation phase (Kasse Initiatives, 2004), as shown in Figure 36. The analysis phase depicts the static representatives of the functional and physical architectures. The first task is to, create a functional representation of an SoS architecture; the second task seeks to identify available systems for alignment to specific functionality needs. On the basis of that, diverse architecture alternatives can be formed and selected. The synthesis phase intends to build relative CPN models with transitions as expression of functions, using the static functional representations as a guideline. The evaluation phase will measure

the performance and cost, which utilizes complexity as an indicator. In fact, SoS might have multiple capability and performance objectives; however, the method in this section simply selects system response time as performance indicator to illustrate the methodology.

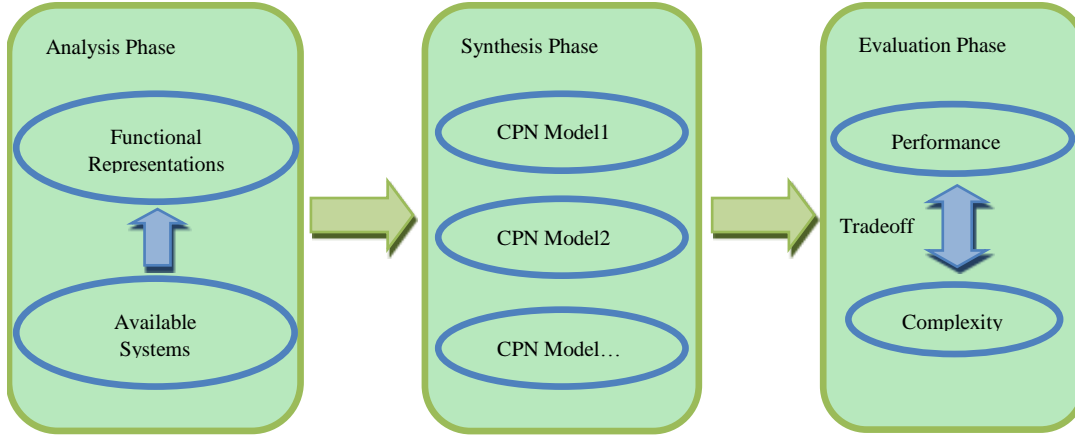


Figure 36. Architecture modeling process

### 3.6.3.2 Complexity Measurement

The overall complexity calculation contains static, functional and concurrency complexity (Fry & DeLaurentis, June 2011), the latter two constituting the dynamic complexity. Since the transitions in a CPN model reflect the functionalities of an SoS architecture, all the three metrics will center on transitions and associated paths. Static complexity stresses the topological structure of architecture. The transitions are divided into two groups motivated by coupling complexity calculation in (Stuart, Mattikalli, DeLaurentis, & Shah, Sept.2011), one is the transition sets comprising transition invariant (T invariant) while the other covers the rest transitions. T invariant indicates a cycle in the state reachability graph, which means that the firing of transitions in T invariant will lead to system returning to the state before the series of firing. The reason for employing T invariant rests on possible added complexity due to functional cycle with certain constraints. Basically this metric counts the total number of input and output places connected to the transition.

$$SCPX = \sum_{k=1}^l st_k \sum_{i=1}^m d_i + \sum_{j=1}^n d_j \quad (32)$$

In the above equation,  $st_k$  represents size of the  $k^{th}$  T invariant,  $d_i$  is the degree of transition  $i$  that in the T invariant and  $d_j$  is the degree of transition  $j$  that is not in the T invariant. Since this metric is based on static structure, it can be computed without firing CPN model.

Functional complexity addresses complexity emerging from dynamic operations. In addition to measuring firing probability of a given transition as Ammar (Ammar, Nikzadeh, & Dugan, June 2001) did, research in this section accounts for the possibility of different scenarios which might lead to different functionalities being executed. For example, in a military context, different backgrounds or situations might result in the employment of completely different strategies. In a warfare scenario, a sensed threat needs to be removed as soon as possible while during peace time, contacts might be required or disruptions will be used instead of destroy. As such, each of these scenarios uses a specific set of functionalities to generate an associated transition path, that will be given a weight provided by experts.

$$FCPX = \sum_{k=1}^n \lambda_k \left( \sum_{i=1}^m (d_i \cdot p_i) \right) \quad (33)$$

where  $\lambda_k$  means the weight of the  $k^{\text{th}}$  path and  $p_i$  is the probability of transition  $i$  being fired, which can be obtained by dividing the number of times each transition is fired by the number of total firings. These firings could be collected automatically during the simulation execution.

Concurrency complexity illustrates another aspect of dynamic behaviors. Concurrent processes, usually associated with resource allocation, occur frequently in SoS operation. A simple example is a sensor system with multiple sensors working parallel to search for a target threat. The added complexity from one sensor performing to multiple sensors working includes not only the new assigned sensors themselves but also the increased interactions and communications between sensors and measures dealing with resource conflicts as well.

$$CCPX = \sum_{k=1}^n \lambda_k \left( \sum_{i=1}^m \left( \frac{ccf_i \cdot d_i \cdot p_i}{f_i} \right) \right) \quad (34)$$

where  $ccf_i$  is the concurrency factor representing maximum number of other transitions enabled concurrently with transition  $i$  across the entire simulation run (Fry & DeLaurentis, June 2011) and  $f_i$  is the number of times transition  $i$  fired.

The static, functional and concurrency complexity are combined to form the overall complexity metric by adding them together based on two assumptions: the inclusion of transition degree in each complexity metric partially eliminates scale differences; the three metrics are equally important and hold tolerable overlap.

$$CPX = SCPX + FCPX + CCPX \quad (35)$$

### 3.6.4 CASE STUDY

The Littoral Combat Ship (LCS) Surface Warfare (SUW) module, one module of LCS naval warfare package, is chosen as a demonstration example. To avoid misunderstanding, “LCS” in this section refers to the LCS seaframe while the LCS package represents the whole naval warfare package involving LCS seaframe. SUW is designed to detect and engage multiple surface contacts in a littoral environment. It strengthens the core LCS seaframe capability by adding an air-to-surface missile armed aircraft and a surface-to-surface missile capability (Jacobson, Sept.2010). Strictly speaking, SUW is not qualified as an SoS, but it nevertheless resembles the SoS attribute that systems are put together to provide capabilities. According to Jacobson’s work (Jacobson, Sept.2010), a basic functional representation of SUW can be created as in Figure 37 and available systems for each function are also shown in the figure. As shown in Figure 37, the functions of SUW used in this example are kept in a high level, from search, to track, to engage, to assess, no more detailed information is used. The abstraction prevents struggling with too many details at the beginning of architecture design process.

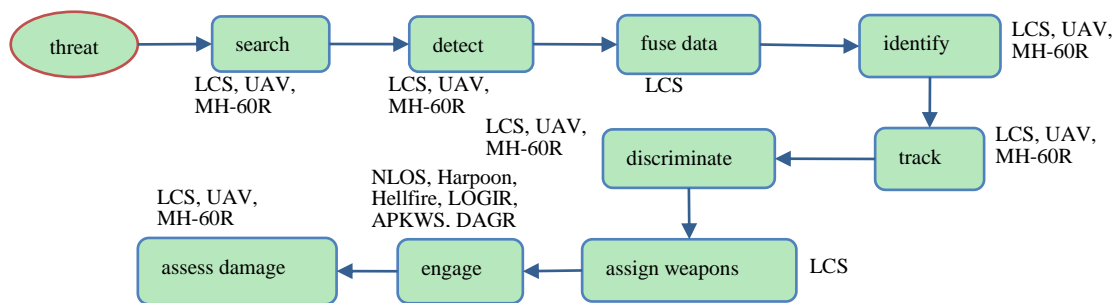


Figure 37. Functional representation of SUW architecture

It is assumed that the current architecture alternative only includes LCS and its NLOS missile system. Two possible evolving architectures are demonstrated in Figure 38, as alternatives to improve performance following the technical dimension previously mentioned. One is equipped with the helicopter MH-60R and its Hellfire missile system, while the other one has both MH-60R and UAV with their respective Hellfire and LOGIR missile systems.



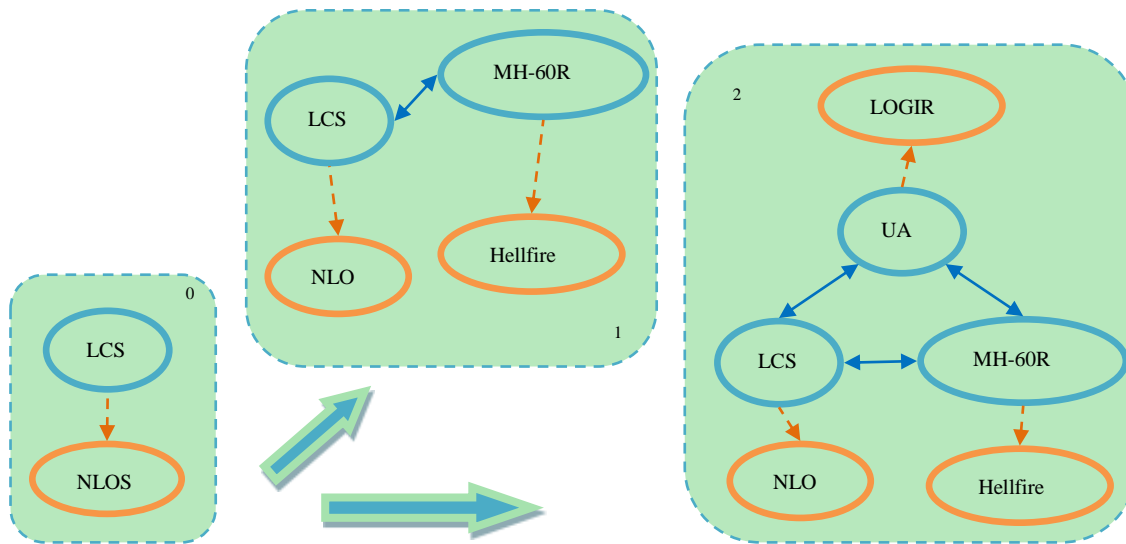


Figure 38. Existing and evolving architecture alternatives

After accomplishing analysis phase of architecture modeling, we are able to proceed to CPN modeling for the dynamic architecture description. Each function in the architecture corresponds to a transition in the CPN. The model of the existing architecture is displayed in Figure 39.

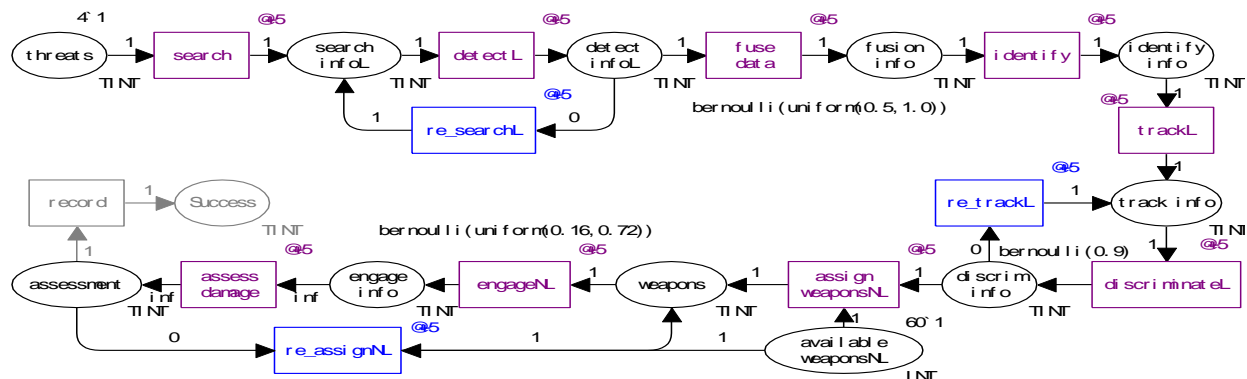


Figure 39. CPN model for existing architecture

System response time is selected as a measure of performance. The model has three primary inputs: 1) process time of each function and 2) detection probability of sensors and kill probability of weapon systems and 3) missile carrying capability for each aircraft. These probabilities and capabilities are based on Jacobson's thesis (Jacobson, Sept.2010). Another input is the number of threats which, if increasing, will lead to increasing system response time. As shown in Table 7 and Figure 40 (performance and complexity with 20 incoming threats), existing architecture0 with LCS solely has the least complexity whereas largest system response time(worst performance); participation of MH-60R reduces the system response time meanwhile increases certain amount of complexity; architecture2 with both MH-60R and UAV apparently has the best performance but maximum complexity. When the number of threats reaches the maximum that SUW can handle, the operations and interactions of systems will not change,

corresponding to a stagnation in complexity and impact in performance. In detail, when there are 20 threats coming together, existing architecture (architecture0) with a complexity value of 66 has average system response time 135.6, while architecture1 has complexity of 90 and system response time of 127.1, comparing to architecture2 with complexity of 118 and system response time of 119.3. Again, while there is no shortage of complexity metrics, the one applied here was selected because of its ease of integration into the framework and its inclusion of dynamic behaviors, but can be replaced with more comprehensive metrics in the future. Additionally, this metric is employed to distinguish the three different architectures. Strictly speaking, the simplest complexity metric as the total number of components should work here. As long as specific requirements, such as required system response time for successful engagement, are given, decision makers can make proper choices on questions like which architecture might be more suitable to the future environment and what kind of evolution is necessary and sufficient based on the tradeoff. With the analysis approach demonstrated, more evolution possibilities can be involved and simulated in the future to improve fidelity of the decision surfaces.

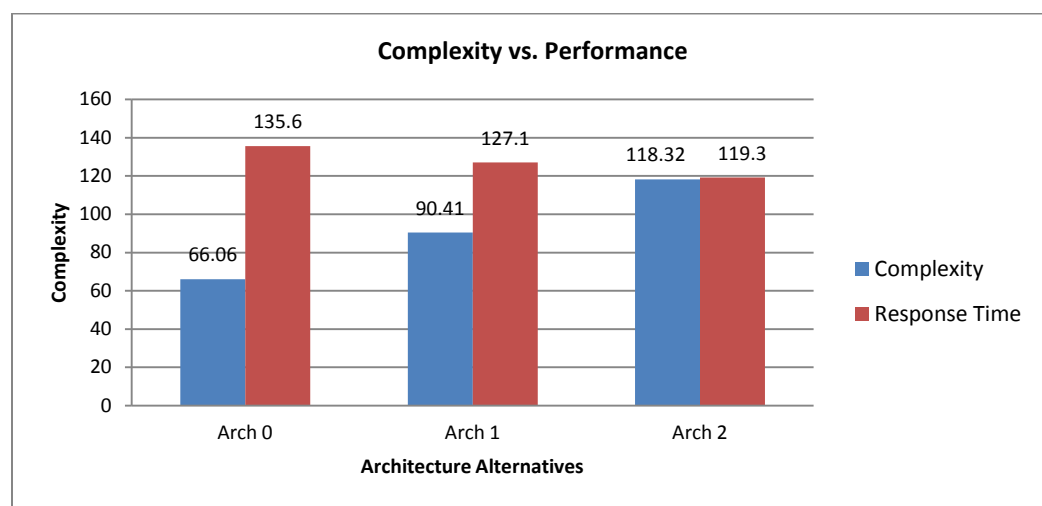


Figure 40. Tradeoff space between performance and complexity

Table 7. System performance and complexity calculation result

Alternatives	Num of threats	Static Complexity	Functional Complexity	Concurrency Complexity	Complexity	Response Time
0	4	34	21.66	14.15	69.81	98.1
	8	34	21.67	16.51	72.18	114.7
	12	34	21.66	17.41	73.07	123.6
	16	34	21.66	13.03	68.69	131.8
	20	34	21.66	10.40	66.06	135.6

1	4	44	22.95	23.44	90.39	78.1
	8	44	22.95	26.83	93.78	94.9
	12	44	22.96	27.61	94.57	108.3
	16	44	22.97	27.76	94.73	120.3
	20	44	22.97	23.44	90.41	127.1
2	4	58	24.21	29.06	111.27	79.9
	8	58	24.21	33.89	116.1	92.7
	12	58	24.21	35.36	117.57	102.3
	16	58	24.20	35.92	118.12	110.4
	20	58	24.20	36.12	118.32	119.3

### 3.6.5 CONCLUSION AND FUTURE WORK

Colored Petri Nets is adopted to construct an executable architectural model due to its ability in modeling dynamic processes and concurrent activities. The corresponding performance and complexity of alternative architectures are calculated to provide a quantitative tradeoff space for decision makers. As a primary contribution, the tradeoff space aids decision makers gaining a comprehensive insight of architecture evolution. A simplified scenario of a SUW module in LCS naval warfare package demonstrates the proposed approach. Results illustrate the increased complexity along with improved performance. Given a baseline of performance and complexity requirements, this approach would provide appropriate suggestions.

The proposed CPN framework is regarded as an initial step (evaluation of architecture alternatives) towards exploring SoS architecture evolution. The ultimate goal for the research is to provide suggestions about when to apply which architecture to support architecture evolution based on such an evaluation. Specifically, short run future work will focus on three aspects. First, is to increase model accuracy; in addition to using Petri net to model high level architecture, agent-based modeling could be adopted to model the lower level of physical architecture. Second, is to increase the efficiency of architecture analysis; on one hand, we will take advantages of Petri net on finding critical events to support architecture evolution; on the other hand, we might employ optimization methods to allow a set of architectures performing in a shorter time to achieve the results. Third direction concentrates on extending complexity metrics by adding more actual information to gradually eliminate the differences between the complexity of models and real systems.

---

## 3.7 ROBUST PORTFOLIO OPTIMIZATION APPROACH TO ARCHITECTING SYSTEM OF SYSTEMS

Current guidelines and tools for architecting SoS architectures are lacking sufficient capabilities in enabling effective decision-making for SoS SE practitioners. This section presents a novel decision analysis framework that draws upon methods from operations research and financial engineering to present a method of architecting a System-of-Systems (SoS) while balancing gains in SoS level capabilities against potential developmental risks. Work in this section extends initial frameworks developed under the Naval Postgraduate School Acquisition Research Program, towards SoS architectural management.

---

### 3.7.1 INTRODUCTION

The recognition of large scale conglomerations of systems as an interacting ‘System-of -Systems (SoS)’ has seen the need for a different set of architecting principles. The Department of Defense (DoD), for example, has changed its requirement centric acquisition process to now reflect a capabilities driven one. The change in paradigm is motivated, in part, by the evolving nature of mission objectives that emphasize for SoS to be adaptable and resilient to changes in mission requirements. These changing conditions present a difficult endeavor in balancing the uncertainty in requirements against the strategic nature of system architectures that are typically static *investments* over a long term horizon. As such, the recognition of managing the complexities of SoS architectures has prompted research in developing tools that can address the management of SoS architectural evolutions.

Recent efforts have attempted to address the development of SoS, by introducing a range of analytical methods for the design of these monolithic systems. Research by (Kotegawa, 2008) utilizes graph theoretic methods with a ‘top-down’ approach to categorize SoS performance and resilience within the context of established network metrics. Works in by (Mane M. C., 2012) on the other hand, adopt an operations research (OR) perspective through concurrent engineering paradigms that seek to improve efficiency by reducing decision ‘handoffs’ between interdependent systems; namely between airline operations and aircraft design. Research by (Mane, DeLaurentis, & Frazho, 2011) pursues a network driven Markov perspective to SoS complexity and attempts to quantify cascading effects of single nodal disruptions on a network. Literature by (Garrett, 2011) has presented works on managing interstitial spaces for a SoS case of the Ballistic Missile Defense System [BMDS]; the work focuses on issues of interoperability, integration and domain interfaces through employment of graph theoretic and agent based methods, coupled with developed managerial and technical metrics.

Although the span of SoS research encompasses both ‘top down’ and ‘bottom up’ strategies, there is still a great need for further development of methods to design and ultimately acquire general SoS constructs under varying conditions due to the span of disciplines that typical SoS constitute. The surveyed methods do not present a decision or ‘portfolio’ based approach that allows for identification of optimal *collections of systems*. This section complements current research efforts in development of SoS architectures by introducing a decision tool via a robust portfolio optimization framework. The generalized representation allows for a SoS to be modeled as an interconnected network hierarchy of

systems with each system being represented by a node. The interconnected capabilities and requirements of these 'nodes' allows for mathematical programming techniques to be used in leveraging performance of SoS under conditions of changing requirements, against associated interconnectivity risks.

---

### 3.7.2 BACKGROUND AND MOTIVATION

Systems engineering has recently benefited from the introduction of operations research driven optimization methods, to developing SoS architectures. Robust optimization is a relatively recent field in operations research, which finds its roots in control theory and addresses the need to make decisions under conditions of uncertainty. Real world systems are subject to inherent uncertainty that manifests itself typically as data and parametric uncertainty. Robust formulations address this inherent uncertainty by generating solutions that are resilient to changes in the data and/or governing parameters of the system. Deterministic methods are traditionally focused on finding the optimal solution of a given optimization problem formulation subject to an assumed set of 'perfect' data. Purely deterministic systems, however, may have very poor response when one or more uncertainties value change. In a SoS, this translates to potentially cascading modes of failure that can propagate throughout the SoS network, resulting in costly degradation in performance, capability and increased cost.

#### *Robust Optimization*

Robust formulation solutions on the other hand, remain near optimal even if the data characteristics change. Robust optimization strategies seek to select appropriate uncertainty sets or alternative parameterizations of the deterministic optimization problem to better deal with robustness issues in the system. Early works by (Dantzig, 1955) have examined formulations for uncertain linear systems of equations. Several formulations have introduced uncertainty in linear integer problems for resource allocation and network flow problems [7]. Further developed formulations in robustness include Soyster, Ben-Tal Nemirovski (Ben-Tal, 1999), and Chance Constrained Programming (CCP) (Charnes, 1959) to name a few. These methods deal with issues on the inclusion and selection of uncertainty sets that transcribe the level and characteristics of uncertainty in the data and/or parameters of the system. These can be polytopic or elliptical sets that are normally used to parameterize uncertainty, resulting in conic and semidefinite optimization problems. Also, the various formulations seek to leverage a finite 'slack' in the system against constraint violations that would result from a given slack budget.

#### *Investment Portfolio: Mean Variance Optimization*

Financial engineering involves the construction of portfolio investment strategies that balances expected return against risk for a basket of potential investment instruments subject to the constraints of available financial resources. A large part of these strategies comes from the construction of portfolios using optimization methods often found in operations research and controls engineering circles. The central idea in portfolio management is the balance of risk versus reward where information on the risk averseness of the investor and volatility in the returns of potential investments are reflected in the portfolio's construction. These measures of volatility are typically generated from past data about the asset (s) of interest. In the context of an SoS, the expected returns correspond to a desired capability from an investment in a system, and the variance (covariance) can be attributed to developmental or operational risks that arise both from the inherent dynamic of the chosen system, and its interactions with connected systems.

Mean variance optimization (MVO) is a much celebrated financial portfolio theory developed by (Markowitz, 1952) that provides a way to select a diverse portfolio of assets in a manner that leverages risk against reward. Several variations of the formulation have appeared since its introduction in the 1950s, but the essential idea is to generate an optimal efficient risk versus rewards frontier. The motivation is to provide an investor with the means to select an optimal portfolio based on the investor's preference of risk tolerance. The general form of the mean variance optimization problem is:

$$\max \left( \mu^T x_i - \lambda (x_i)^T \Sigma_{ij} (x_j) \right) \quad (36)$$

subject to:

$$\sum_{i=1}^N x_i = 1 \quad (37)$$

$$x_i \geq 0 \quad (38)$$

The mathematical model shown by Equations (36–38) presents the formulation of a traditional single-stage optimization problem that is typical of operations research and financial engineering circles. Here,  $x_i$  is the proportion of funds invested in asset  $i$ ,  $\mu_i$  is the expected return for fund  $i$  and  $\lambda$  is the risk averseness constant measure.  $\Sigma_{ij}$  is the symmetric covariance matrix that contains information on how asset pricing moves together. This matrix is positive definite since variance is always a positive quantity making it a quadratic programming (QP) problem, for which very efficient solution methods exist. Equation (36) is the objective function that seeks to maximize the expected return of the portfolio and minimize its risk. Equation (37) simply ensures that the portfolio fractions sum to a whole and Equation (38) enforces a non-negativity condition of the decision variables - indicative of a no shorting policy.

### *Robust Mean Variance Optimization*

It is well known in financial engineering circles that the Markowitz formulation is sensitive to changes in estimated entries of the covariance matrix,  $\Sigma$ , (system interdependencies) and expected return,  $\mu$ , (system performance). The sensitivity due to suboptimal expected returns and covariance estimations can result in highly inefficient portfolios due to errors in estimation or market shifts. Such sensitivity issues have prompted the development of a variety of robust methods in portfolio analysis to ensure that the chosen portfolio of assets are stable against potential changes in market conditions/expected volatility.

The SoS investment portfolio formulation in Equations (36-38) can be made robust through use of a range of robust optimization techniques. Recent research has adopted robust optimization approaches such as Semi-Definite Programming (SDP) and Conic Programming approaches, among many others, to generate robust portfolios that are protected against uncertainties in data and relevant estimated quantities. The reformulation allows for possible changes in estimated quantities (e.g., due to market shifts in pricing, volatility, system interdependencies) to be accounted more explicitly as uncertainty sets. The resulting portfolio allocation will not change appreciably even if estimated quantities of risk or payoffs change (within prescribed limits).

In the context of an SoS problem, the use of a robust formulation translates to reduced costs associated with capability estimation errors, development time volatility, and changing requirement conditions. In this research, we adapt a Semi-Definite Programming (SDP) approach as developed by Tutuncu and Koenig (Tutuncu, 2004) that addresses general uncertainty in the expected returns, and covariance matrix,  $\Sigma$ , and the Bertsimas-Sim (Bertsimas, 2004) approach to dealing with uncertainties in particular linear constraints. Work by Tutuncu and Koenig (Tutuncu, 2004) present the following robust formulation for the mean-variance portfolio problem:

$$\begin{aligned} & \text{maximize} && \left\{ \sum_q \mu_q w_q + \delta_i (w_+ + w_-) - \lambda \left\{ \langle \bar{\Lambda} \bar{\Sigma} \rangle - \langle \underline{\Lambda} \underline{\Sigma} \rangle \right\} \right\} \\ & w, w_+, w_-, \bar{\Lambda}, \underline{\Lambda} && \end{aligned} \quad (39)$$

subject to:

$$\sum_i w_i = 1 \quad (40)$$

$$\begin{bmatrix} \bar{\Lambda} - \underline{\Lambda} & w \\ w' & 1 \end{bmatrix} \succeq 0 \quad (41)$$

$$w = w_+ - w_-, w_+ \geq 0, w_- \geq 0 \quad (42)$$

$$\bar{\Lambda}, \geq 0, \underline{\Lambda} \geq 0 \quad (43)$$

### *Robust Constraints: The Bertsimas-Sim Approach*

The robust portfolio method shown above address uncertainty in the covariance matrix and expected returns but not in general linear constraints. The Bertsimas-Sim method on the other hand is a robust linear formulation that addresses parametric data uncertainty in linear constraints without excessively penalizing the objective function. The method allows for the control of probability of constraint violations and the effect of the degree of conservatism on the objective function, also known as the price of robustness. Its linear formulation makes it naturally extendable to discrete optimization problems and is a very attractive method for application to the current (linear) SoS architectural framework. The formulation starts by addressing the general inequality constraints in a traditional linear programming problem;  $Ax \leq b$ . Here, a subset of matrix  $A_{ij}$  contains uncertain entries (derived from data) that exist within symmetric intervals – i.e. belong to set  $J_i$  and  $[a_{ij} - \hat{a}_{ij}, a_{ij} + \hat{a}_{ij}]$ . A conservatism parameter,  $\Gamma_i$ , is used to adjust the degree of conservatism in protection of the uncertain linear constraints from infeasibility,  $\Gamma_i$  takes values in the interval  $[0, |J_i|]$  where  $J_i$  represents the set of  $A_{ij}$  coefficients that are uncertain and is not necessarily integer. Introduction of uncertainty in the coefficients results in a nonlinear problem formulation. However, a proof is derived in literature [10], that converts the nonlinear form into the following linear optimization problem:

$$\text{maximize} \quad c^T x \quad (44)$$

subject to:

$$\sum_j A_{ij} x_j + z_i \Gamma_i + \sum_{j \in J_i} p_{ij} \leq b_i \quad (45)$$

$$z_i + p_{ij} \geq \hat{a}_{ij} y_j \quad (46)$$

$$-y_j \leq x_j \leq y_j \quad (47)$$

$$l_j \leq x_j \leq y_j \quad (48)$$



$$p_{ij}, y_{ij}, z_{ij} \geq 0 \quad (49)$$

The linear formulation shown in the above equations preserves sparsity of the original  $A_{ij}$  matrix - an attractive feature for computational efficiency. Equation (44) is the objective function that maximizes a general linear function. Equations (45-49) are the robust version of linear inequality constraints where  $\Gamma_i$  is the constant that dictates the level of conservatism in the constraint.  $\hat{a}_{ij}$  in Equation (45) is the uncertainty associated with the j-th entry of the i-th constraint in the A matrix. The SoS architectural formulation, as will be presented, provides naturally sparse systems of equations and complements sparsity advantages in the Bertsimas-Sim formulation. In general, linear formulations are amenable to highly efficient solvers that can handle very large scale problems with ease.

Various robust formulations attempt to address uncertainty through Monte Carlo sampling, conic programming and semidefinite programming approaches - these are more computationally expensive to solve than the linear formulation shown above. The Bertsimas-Sim formulation as shown, does not take correlation effects into consideration. Here, the uncertainties are assumed to be independent sets that exist separately from one another. However, work developed by Bertsimas and Sim also includes alternate (linear) formulations that extend the formulation also account for correlated data. This naturally makes it applicable to an even wider range of problems - including asset allocation and investment problems such as the mean-variance portfolio optimization problem.

### 3.7.3 ROBUST PORTFOLIO APPROACH: SoS NETWORK MODELING

In this research, a generic, operational SoS architecture is modelled as an interconnected set of discrete nodes; each having a finite set of inputs and outputs. The interconnectivities between nodes are established to facilitate the fulfilment of individual node requirements by allowing for node capabilities (outputs) from existing nodes to connect and consequently fulfil requirements (inputs) of any compatible node requiring a particular capability to function. Overarching capabilities are provided by nodes that directly contribute to these required capabilities.

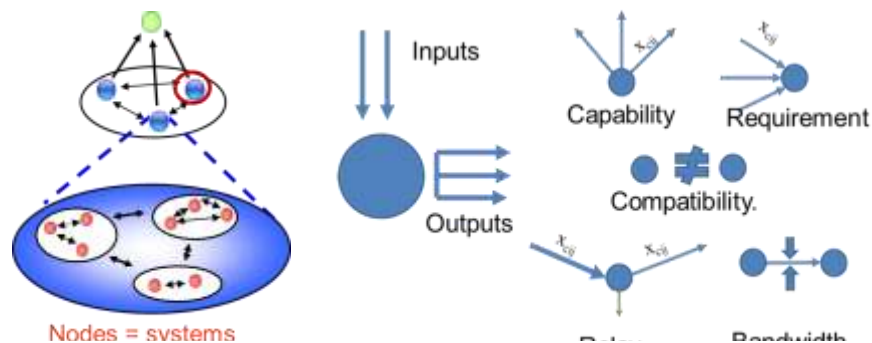


Figure 41: (a) SoS hierarchy (b) nodal behaviors

Figure 40 (a & b) show a generalized representation of a SoS which has interdependencies between constituent systems, across multiple layers of the hierarchical structure. The hierarchical structure transcends the operational layers of the SoS construct. Each node (system) is connected to other nodes on the network, in accordance with the set of requirements needed for them to interdependently operate. The connections between nodes are also governed by a set of interaction rules. The idea is to model basic, aggregate interactions between systems within a SoS construct as relatively simple, nodal behaviors that are applicable to a wide variety of types of inter-system connections. While not exhaustive, the combinations of these nodal behaviors as modeling rules can cover a large set of real world inter-system interactions. This is in line with insights that an SoS SE practitioner may have when architecting complex, larger scale systems at a higher architectural level.

Figure (40b) shows the five most intuitive nodal interactions of:

- *Capability*: Nodes have finite supply of capabilities that are limited by quantity and number of connections.
- *Requirements*: Nodes have requirements to enable inherent capabilities. Requirements are fulfilled by receiving connections from other nodes that possess a capability to fulfill said requirements.
- *Relay*: Nodes can have the ability to relay capabilities between adjacent nodes. This can include excess input of capabilities that are used to fulfill node requirements.
- *Bandwidth*: Total amount of capabilities and number of connections between nodes are bounded by 'bandwidth' of the connection linkages between systems.
- *Compatibility*: Nodes can only connect to other nodes based on a pre-established set of connection rules.

The performance of the SoS is related to the ability of the connected network of individual systems to fulfill overarching core objectives. The SoS wide performance is quantified by the capability of nodes that most directly contribute to the core objectives. It is assumed that these core objectives can be at least, approximated quantitatively.

### *Robust Investment Portfolio for SoS*

The SoS investment model is posed as a mean variance investment problem. The objective here is to maximize the expected network performance in fulfilling key overarching objectives whilst minimizing acceptable levels of developmental risk and cost. Selection of system is constrained by fulfilling generic connectivity requirements between constituent systems, as defined earlier. The resulting mathematical program for the SoS investment problem is given by the following equations:

$$\max \left( \sum_q \left( \frac{S_{q \in c} - R_c}{R_c} \cdot w_c \cdot x_{q \in c}^B \right) - \lambda \left( x_q^F \right)^T \Sigma_{ij} \left( x_q^F \right) - \sum_q \left( C_q x_q^B \right) \right) \quad (50)$$

subject to:

$$\sum_i x_{cij} \geq x_j^B S_{rj} \quad (51)$$

$$\sum_i x_{cij} \geq x_j^B S_{rj} \quad (52)$$

$$x_1 + L + x_n = 0 \quad (53)$$

$$\sum_c x_{cij} - x_{ij} M \leq 0 \quad (54)$$

$$M \sum_c x_{cij} - x_{ij} \geq 0 \quad (55)$$

$$\sum_j x_{ij} \leq \text{Limit}_i \quad (56)$$

$$\sum_i x_{cij} - \sum_j x_{cij} - x_j^B S_{rj} = 0 \quad (57)$$

$$x_{cij} \leq \text{Limit}_{cij} \quad (58)$$

$$x_q^F = \frac{x_q^B C_q}{\text{Budget}} \quad (59)$$

$$\sum_q (C_q x_q^B) \leq \text{Budget} \quad (60)$$

$$x_{ij}, x_j^B \in \text{binary } \{0,1\} \quad (61)$$

$$x_{cij}, x_q^F \in \{\text{integer, real}\} \quad (62)$$

The current form for the investment model as shown in Equations (50-62) is known as a Mixed Integer Quadratic Program (MIQP) and is based on the mean variance formulation that seeks to generate optimal portfolios that balance potential expected rewards against risk. Equation (50) is the objective function that seeks to maximize overall SoS capability while minimizing cost and development risk. The first term in the objective function normalizes the ‘capabilities’ by dividing the potential capability of available systems,  $S_q$ , by a baseline performance,  $R_c$ . The capability gain here assumes a linear form. A weighting factor,  $w_c$ , is used to weight the importance of each capability that contributes directly to the objective function. This vector of weighting factors can be determined through design consensus as to prioritization of capability priorities.

Risk is captured through the variance term and moderated by the risk aversion factor,  $\lambda$  that reflects the investor’s risk averseness preference. The negative sign indicates a penalization of the objective function, given the selection of individual systems in the SoS portfolio of systems, and attempts to balance the gains from selecting individual systems with the accumulated risks from them. An alternate formulation of introducing risks involves the inclusion of the quadratic risk term in the constraints as  $x^T \Sigma x \leq \sigma$ , instead of in the objective function, where  $\sigma$  is the total variance (risk) in the portfolio. In the context of a SoS investment framework, the risk here refers to project developmental time of and between interconnected systems; the risk in delays can be related to extra work-hours and thus be quantified as an explicit cost. For new, yet to be designed systems, estimations of the distribution curve for development time are done by design intuition and follow traditional project management guidelines (e.g. PERT/CPM) in systems engineering as listed in literature

Equation (51) is the capacity limit at each node. This ensures that the individual capabilities at each node do not provide more capability to other parts of the SoS network than what is available at the respective node. For example, the selection of a system with communications capabilities should mean that the total bandwidth of data that is being used by other interfacing systems should not exceed the inherent limits of the communications system itself. Equation (52) ensures that requirement conditions are met for each node and permits the availability of excess capability to be present at the node. Equation (53) enforces compatibility constraints in the form of binary logic. For example, if some of the systems represent the choice of engine system to a car; it is not feasible that two 200hp engines may be selected to fulfill the requirements of 400hp. Instead, a restriction is placed such that only one engine may be selected and that the chosen engine must supply 400hp. The engine constraints would thus be written as  $x^{\text{engine1}} + x^{\text{engine2}} = 1$ ; since the decision variables are binary, there can be only a selection of one or the other for the constraint to be satisfied.

The restrictions for a SoS architecture needs user defined inputs on acceptable connectivity constraints in the form of number of connection between nodes. Equations (54-56) are associated with connection restrictions between nodes that follows a ‘Big-M’ formulation. The Big-M formulation establishes a logic condition where if the entry of  $x_{cij}$  is non-zero, then the corresponding entry of  $x_{ij}$  must be nonzero (in this case, a value of 1). The Big-M thus allows for the tracking of connections between systems and permits control of the number of connections to individual systems as well through the adjacency variable  $x_{ij}$ . The summation in Equation (56) enforces the maximum allowable

number of links for each selected system. Equation (57) is a flow balance equation that ensures that 'capability flows' about the interconnected network of systems are preserved (e.g. power, communication connectivity and bandwidth between systems). Equation (59) is the budget fraction equation quantified the fraction of total budget that each invested system represents. Equation (60) ensures that the total cost of systems acquired is within the prescribed budget. It is assumed that the budget is a known and fixed quantity. Equation (61) and (62) show that the decision variables can be integers, binary or real numbers and is dependent on the individual system properties.

---

### 3.7.4 EXAMPLE CASE STUDY: LITTORAL COMBAT SHIP

The Littoral Combat Ship (LCS), shown in Fig 40, is a current system that is both developed by Lockheed Martin and General Dynamics for the United States Navy. The operations of the ship support littoral warfare with a new paradigm in naval architecture that utilizes modular systems. Each module corresponds to a particular set of capabilities that can be added and removed for each ship, depending on the requirements of the mission to be performed. The current suite of modules includes the Anti-Submarine Warfare (ASW), Mine Warfare (MIW), Surface Warfare (SUW) and irregular warfare module for medical/humanitarian based missions. The LCS works in concert with other military entities that include UAVs, satellite systems and ground based units. Whilst the LCS platform is not strictly speaking, a SoS, it exhibits many salient features of one. The modularity and open connectivity allow for it to be redeployed with various collections of assets to fulfill an overarching capability.



Figure 42: Littoral Combat Ship (LCS)

#### *Robust Investment Portfolio Approach*

The objective in this section is to achieve desired combat effectiveness and operational capabilities while minimizing cost and development risk; this is of course subject to connectivity constraints based on the generic nodal behaviors as listed in the preceding section. The simple LCS model inputs and

characteristics are described in Table 8 and represent a hypothetical catalogue of systems available to the Navy in its pursuit of achieving desired capabilities. Although the numbers are hypothetical, the salient features of considering capabilities, requirements and considering risk in acquisition problem are still preserved. Table 8 lists systems for each of the three mission packages—ASW, MCM, SUW—along with an individual rating of system capabilities and requirements for the systems to operate. Additionally, Table 8 provides the system development time and associated acquisition costs. Systems that are unable to provide a particular capability (or do not have a particular requirement) have a zero entry. The development of these systems is based on a estimated time schedules that typically follow PERT/CPM practices in estimations of the distribution of development times. This aspect is captured in the covariance matrix shown in Table 9.

Table 8: LCS candidate systems

		System Capabilities					System Req.	Develop. Time	Acq. Cost	
		Weapon Strike Range	Threat Detection Range	Anti Mine Detection Speed	Comm. Capacity	Air/Sea State Capacity	Air/Sea State	Comm. (Years)	(\$)	
Package										
ASW	Variable Depth	0	50	0	0	0	0	250	3	3000000
	Multi Fcn Tow	0	40	0	0	0	0	150	2	2000000
	Lightweight tow	0	30	0	0	0	0	100	4	4000000
MCN	RAMCS II	0	0	40	0	0	3	200	1	1000000
	ALMDS (MH-60)	0	0	30	0	0	4	100	2	2000000
SUW	N-LOS Missiles	25	0	0	0	0	0	200	3	3000000
	Griffin Missiles	3	0	0	0	0	0	100	4	4000000
Seaframe	ackage System 1	0	0	0	400	4	0	0	3	3000000
& Combat	ackage System 2	0	0	0	300	4	0	0	4	4000000
Management	ackage System 3	0	0	0	250	3	0	0	5	5000000

Table 9: Development covariance matrix

	Variable Depth	Multi Fcn Tow	Lightweight tow	RAMCS II	ALMDS (MH-60)	N-LOS Missiles	Griffin Missiles	Package System 1	Package System 2	Package System 3
Variable Depth	0.1	0	0	0	0	0	0	0	0	0
Multi Fcn Tow	0	0.6	0	0	0	0	0	0	0.1	0
Lightweight tow	0	0	0.2	0	0	0	0	0	0	0.2
RAMCS II	0	0	0	0.3	0.1	0	0	0	0.2	0
ALMDS (MH-60)	0	0	0	1	0.1	0	0	0	0	0.3
N-LOS Missiles	0	0	0	0	0	0.5	0.2	0	0.1	0
Griffin Missiles	0	0	0	0	0	0.2	0.3	0	0	0
Package System 1	0	0	0	0	0	0	0	0.5	0	0
Package System 2	0	0.1	0	0.2	0	0.1	0	0	0.3	0
Package System 3	0	0	0.2	0	0.3	0	0	0	0	0.2

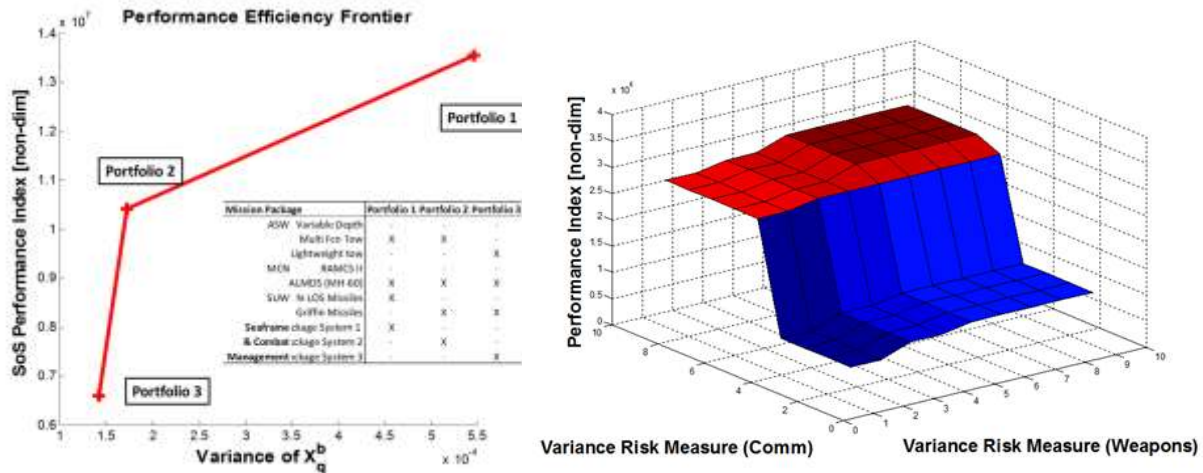


Figure 43 (a) Robust efficiency frontier (b) Multiple risk measures

The robust portfolio optimization formulation of Equations (50-62) is applied and solved for the investment problem for the LCS platform, using data from Table 8 and Table 9, resulting in the robust efficiency frontier and corresponding optimal portfolio of systems at each level of risk aversion,  $\lambda$ . At high levels of risk aversion (high  $\lambda$  values), the portfolio selection favors less risky (lower developmental variance) systems, but at a lower SoS wide capability level and vice versa. The method can be extended to the case of multiple measures of risk as shown in Figure 42(b). Here, two measures of risk are introduced to the optimization framework; the first reflects risks in the communications layer and the second, in the weapons effectiveness. Both measures are represented as covariance matrices with hypothetical entries,; however, these can conceivably be retrieved from quantitative measure such as field test/simulation data. In Figure 42 (b), the dual measure of risk here reveals a trade-space topology where increasing risk aversion in either measure results in increased SoS wide performance. The discrete jumps in performance are attributed to the nature of the systems that are discretely connected to one another.

#### Bertsimas-Sim Approach to Robust Operational Constraints

The objective in this part of the synthetic case application is to identify collections or ‘portfolios’ of systems, chosen from a candidate list, that fulfill and overarching SoS capability set (ASW,MIW,SUW) whilst satisfying probabilistic constraints for specific layers of the connections between systems. Table 8 presents extended candidate system information for the Littoral Combat Ship (LCS) operational network. Each candidate system has a collection of capabilities and requirements as listed; in this case, additional systems are available for the communications layer which is the focus of probabilistic constraints in this section of the research. The individual system capabilities, as listed in columns 1-5, can be used to either directly fulfill an overarching SoS requirement (listed in columns 1-3), or to fulfill individual support system requirements (columns 4-9). Columns 6-7 are systems requirement metrics across the candidate



systems. Zero value entries in these columns indicate that the respective listed system does not have that particular system requirement to be fulfilled. In this simplified scenario, it is assumed that a communications layer exists where all assets in Table 1 have an ability to ‘communicate’ with one another in the transfer of information, subject to a path-wise cost. The objective here is to select a collection of assets (system) from the available list in Table 1 to that maximizes mission performance requirements; this comprises an equal weighting among primary SoS capabilities of the first 3 columns for the LCS problem. Additional constraints include the fact that only one system can be selected for each package with the exception of the communications packages where a total of up to two may be deployed.

Table 10: LCS candidate systems

Package		Weapon Range	Detect. Range	Anti Mine	Comm. Capability	Power Capability	Power Require.	Comm. Require.	Uncertainty Set
ASW	Variable Depth	0	50	0	0	0	100	200	0
	Multi Fcn Tow	0	40	0	0	0	90	120	0
	Lightweight tow	0	30	0	0	0	75	100	0
MCN	RAMCS II	0	0	10	0	0	70	120	0
	ALMDS (MH-60)	0	0	20	0	0	90	150	0
SUW	N-LOS Missiles	25	0	0	0	0	0	250	0
	Griffin Missiles	3	0	0	0	0	0	100	0
Seaframe	Package 1	0	0	0	0	300	0	0	10
	Package 2	0	0	0	0	450	0	0	70
	Package 3	0	0	0	0	500	0	0	100
Comm.	System 1	0	40	0	180	0	100	0	30
	System 2	0	200	0	200	0	120	0	35
	System 3	0	0	0	240	0	140	0	45
	System 4	0	0	0	300	0	160	0	55
	System 5	0	0	0	360	0	180	0	60
	System 6	0	0	0	380	0	200	0	80

The selection is subject to the performance and associated uncertainty that exists in the communications network of the selected assets. Uncertainty exists in the communications and power capabilities of individual systems as published in columns 4 and 5 of Table 8. The values for communications bandwidth (column 4) and power generation (column 5) are nominal values,  $a$ , that are subject to uncertainties  $\hat{a}$  to form the uncertainty set of  $[a_{ij} - \hat{a}_{ij}, a_{ij} + \hat{a}_{ij}]$ . The uncertainty bounds, in real applications, can typically be estimated from designer intuition, simulation outcomes or confidence intervals from reported performance data. The sufficiency conditions in Equations (8) and (9) of the optimization formulation are adapted to their robust form using the Bertsimas-Sim formulation. The purpose here is to evaluate the probability of sufficiency constraint violation, given differing degrees of built in robustness, using the conservatism term of the robust formulation. The robustification is only applied to constraints of Equations (8) and (9), but can be applied to any other linear constraints in a general MIP problem.

Figure 43 (a) and (b) show the SoS performance profile and ‘portfolio’ of selected assets respectively for the simple LCS scenario. The generation of the profile comes from the solution of the optimization problem as defined in Equations (7-16) for the LCS set of data, using varying conservatism values of  $\Gamma_i$ .

The problems were solved in the MATLAB [12] environment using YALMIP [13] interface with the Gurobi solver option. The simple degree of complexity in the problem results in three distinct architectures as shown in Figure 4 (a) and (b). The graph shows the increase in SoS performance index with a decrease in the level of conservatism,  $\Gamma_i$ . This is as expected, given that the increase in robustness will yield a trade-off between performance and conservatism. The three communications architectures have different asset selections that are reflected in the MCN, SUW and choice of communications systems. As the degree of conservatism in the SoS communications capability is reduced, the resulting system selection allows for the higher capacity communications system 6 to be selected, even though it bears a greater degree of uncertainty. This in turn allows for the selection of NLOS missiles but degrades the MCN performance to the selection of the RAMCS II system instead of using the ALMDS (MH-60) unit. Further reduction in conservatism results in the NLOS and ALMDS units to be selected, maximizing the respective SoS level capabilities for each package category. The choice of conservatism is typically set by the SoSE practitioner that may include consideration of acquisition cost. Here, the probabilistic guarantees at discrete levels of conservatism can be weighed against the potential acquisition costs or against the potential of further upgrades to the SoS architecture. The small levels of probabilistic increments are artifacts of the synthetic numbers used in this simplified problem.

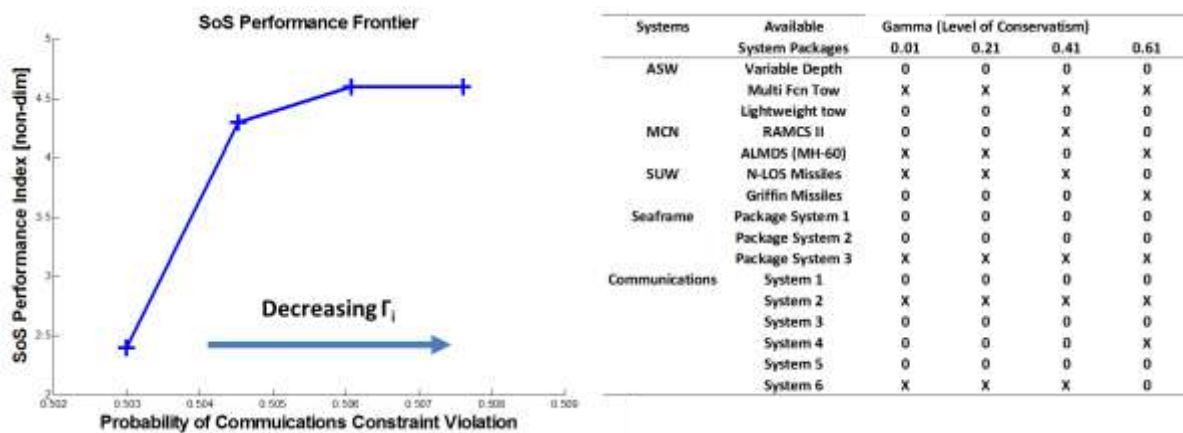


Figure 44: (a) performance profile (b) portfolio of assets








### 3.7.5 SUMMARY AND FUTURE WORK

A robust portfolio optimization approach is presented that includes consideration for uncertainties in various aspects of the resulting SoS ‘investment’ problem. The method is applied to the case of a simplified LCS problem to reflect potential decisions that SoS SE practitioners can make in view of generated trade space efficiency frontiers that balance risk against capability and cost. The method includes consideration of the interconnectivities between systems, access to multiple measures of risk and considerations for probabilistic guarantees on connectivity.

Future work in this endeavor will seek to better unify the Bertsimas-Sim and robust portfolio approaches within a single measure and apply the methods for multi-period portfolios where decisions of architecting in one time state can affect subsequent SoS architectural decisions.

## 4 SUMMARY AND FUTURE RESEARCH

This report has detailed the results of research, development and application of MPTs in support of SoS architectural artifacts. The methods enable the analysis and support of system capability development using defined metrics and provide a means to conduct analysis of alternatives while navigating the SoS decision space. These methods also simultaneously enable the identification of positive impacts of interdependencies (SoS level capabilities and resilience) while mitigating negative one (e.g. developmental delay risks). The methods are demonstrated for simplified and instructive test cases and have shown great promise in providing ‘value-added’ to SoS SE practitioners in navigating SoS level design spaces, and performing measurable actions that translate to risk mitigation and capability development.

Methods	Nature	Inputs/outputs	Most Relevant Action
FDNA/DDNA	Topological Graph	Connectivities, SOD, COD	
Stand-in Redundancy	Probabilistic Graphical	Probabilities, Connectivities	 
Bayesian Network (BN)	Probabilistic Graphical	Conditional Distribution Connectivity	
Color Petri Net (CPN)	Discrete Event Simulation	Connectivities, Transition rules	
Robust Portfolio Approach	Decision/Analysis based	Capabilities, Requirements, Connection rules	 

*Future Research: Towards an Analytical Workbench*

The table presented above highlights the nature of each explored method and the required input metrics for applicability to particular analytical situations. The table also notes the most pertinent SoS artifacts, for as defined in the Wave Model, for each method. While the listed method(s) recommended for each relevant SoS artifact action is listed, the methods are nevertheless still applicable to other artifacts in other SoS artifacts, but is naturally dependent on the nature of the problem, and the

pertinence of the inputs/outputs as listed in the above table. For example, the FDNA/DDNA method, while structured for higher level SoS analysis functions, can equally be used to analyze consequences of network connectivities as implied for Colored Petri Nets should the pertinent data be available.

The research conducted in this RT-36 epoch has accomplished the following details:

1. Categorize types/styles of SoS architectures, metrics for SoS performance and development, and interdependency features that must be addressed by analysis methods. This included literature review with SERC projects (e.g., RT18 and RT25) but also more widely.
2. Define and translate the meaning of metrics for the SoS context and determine which metrics can be computed, and under which assumptions, for each candidate analysis method, in relation to the SoS artifacts in the Wave Model
3. Developed instructive synthetic examples to test the analysis methods; also, look for real case studies to use. We completed demonstration studies on a 5-node network sample problem and/or a Littoral Combat Ship scenario using five methods for interdependency analysis: Markov-network process, Bayesian Network, Petri Nets, 'stand in redundancy' reliability . A decision support method using robust portfolio optimization is also presented. We documented the workings of each method on the problem and reported. We also created our own case study and example problem based on the Littoral Combat Ship (LCS) program.
4. Completed test cases that highlighted strengths and weaknesses of each method and couched the products of each method in context of the Wave Model for SoSE.

Additional research under RT-44 that continues research here in RT-36 will be centered on developing an 'analytic workbench' and advancing/refining the current range of methods to better address interdependencies in SoS SE architecting. The workbench will consist of software tools that are, possibly, modular so as to enable addition of new features. The modularity will include consideration for an open-source environment and thus requires some adaptation of current researched methods to support relevant open-source software considerations. The methods support the Wave Model artifacts for SoS architecture proliferation as shown in the following:

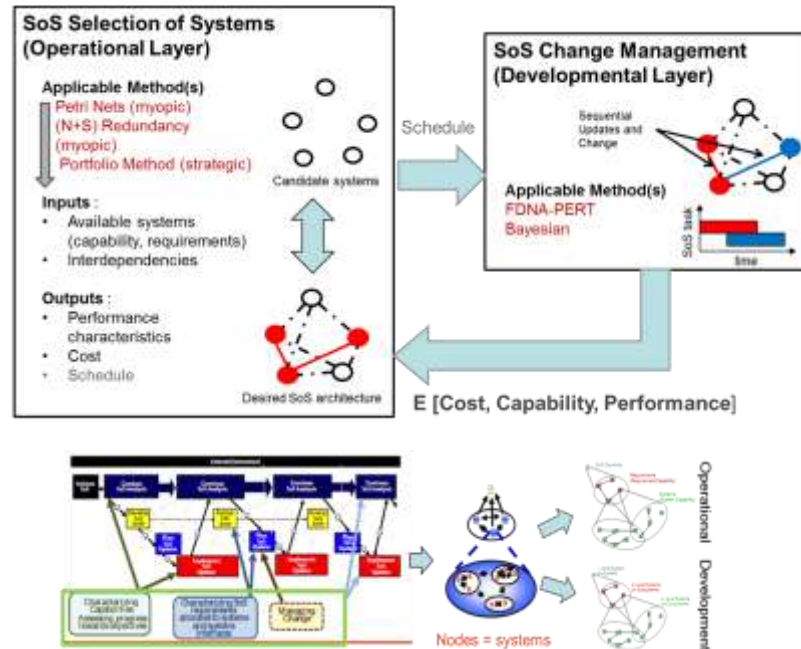


Figure 45: SoS and Wave Model evolution

The above Figure 1 illustrates the researched methods in a generalized context of supporting SoS architecture evolution. The analytical workbench would serve in the application of prior researched methods in supporting *operational* and *developmental* hierarchies, in accordance with the Wave Model. This includes decision-making involving metrics that reflect trade off among ‘illities’ of the SoS entity (reliability, resilience...etc.) as part of the generalized SoS capability. The concept framework shown in Figure 1 also highlights the sequential application of methods that cover the range of SoS activities; this spans from the strategic evaluation of individual systems to the tactical aspects of change management and development. The cyclic nature of the shown framework reflects artifacts of the ‘Vee’ structures in the Wave model.

## 4.1 WHAT CAPABILITIES WILL FUTURE RESEARCH BRING?

Research during the period would also result in papers and reports that document experimentation with the tools on realistic test problems, with a variety of data availability and use environments. The analytic workbench (tools and studies) would be oriented to help practitioners to answer questions such as:

- How are vulnerabilities identified and traced to architecture features and interdependence structure?
- What can be done (new systems, transitioned architecture) to maximize my capability gain (or resilience)?
- How to understand the alternatives in cognizance of cascading effects and dynamics?

- How can investments now address current needs but also longer term evolution goals?

---

#### 4.2 ANALYTIC WORKBENCH: VERIFICATION, DEMONSTRATION AND VALIDATION

The additional research would apply the researched methods and established framework above to a more realistic concept problem for validation; this involves a higher fidelity definition of the simple concept Littoral Combat Ship (LCS) scenario that was used in preliminary works. Our goal would be to further enhance *value added* aspects of the proposed workbench by illustrating its benefits to SoS practitioners through example applications of the workbench in supporting SoS evolution of the LCS program, as a case study. More specifically, the demonstration, validation and verification would accomplish the following:

:

- Develop a high fidelity concept problem based on the LCS platform as a SoS problem. This involves an Agent-Based Model that acts as a simulation test bed for operational decisions.
- Define metrics for the LCS problem in a SoS context and determine which metrics can be computed, and under which assumptions, for each candidate analysis method.
- Demonstrate, within the structure of an analytical workbench, the application of researched methods in architecting and evolving the LCS SoS construct. This involves application of each method to representative architectural challenges that SoS practitioners may face in developing the LCS problem.
- Validation and verification of efficacy of methods in the context of the extended SoS problem (e.g. LCS problem).

## 5 BIBLIOGRAPHY

---

- LCS: *The USA's Littoral Combat Ships*. (2012, March 18). (Defense Industry Daily) Retrieved March 18, 2012, from <http://www.defenseindustrydaily.com/the-usas-new-littoral-combat-ships-updated-01343/>
- Ammar, H., Nikzadeh, T., & Dugan, J. (June 2001). Risk Assessment of Software System Specifications. *IEEE Transactions on Reliability*, Vol.50(No.2), pp:171-183.
- Arteta, B., & Giachetti, R. (May 2004). A Measure of Agility as the Complexity of the Enterprise System. *Robotics and Computer-Integrated Manufacturing*, Vol.20, pp:495-503.
- Avallone, E., Baumeister, T., & Sadegh, A. (2007). *Marks' Standard Handbook for Mechanical Engineers 11th Edition*. New York: Mc-Graw Hill.
- Ayyalasomayajula, S. (May 2011). *A FORMULATION TO ANALYZE SYSTEM-OF-SYSTEMS PROBLEMS: A CASE STUDY OF AIRPORT METROPLEX OPERATIONS*. West Lafayette, IN: PhD Dissertation, School of Aeronautics and Astronautics, Purdue University.
- Ben-TalNemirovski, A.,A.,. (1999). Robust solutions of uncertain linear programs. "Operations Research Letters", 25, 1-13.
- BertsimasSim,M.,D.,. (2004). The Price of Robustness. "Operations Research", 52(1), 35-53.
- Blanchard, B., & Fabrycky, W. (1998). *System Engineering and Analysis, 3rd ed*. Prentice Hall International Series in Industrial and Systems Engineering.
- CharnesCooper, W.,A.,. (1959). Chance-Constrained Programming. "Management Science", 6(1), 73-79.
- Crossley, W. A. (2004). *System of Systems: An Introduction of Purdue University Schools of Engineering's Signature Area*. Cambridge: presented at the Engineering Systems Symposium, MIT Engineering Systems Division.
- Czapiewski, P. (2004, Sept. 23). *Littoal Combat Ship*. Retrieved Dec. 21, 2011, from [http://www.foils.org/01\\_Mtg\\_Pres%20dnloads/LCS\\_SNAME\\_IHS041023.pdf](http://www.foils.org/01_Mtg_Pres%20dnloads/LCS_SNAME_IHS041023.pdf)
- Dahmann, J., Rebovich, G., & Lowry, R. (May 2011). An Implementers' View of System Engineering for System of Systems. *Proceedings of Institute of Electrical and Electronics Engineers Systems Conference*. Vancouver, Canada.
- DantzigGeorge. (1955). Linear programming under uncertainty. "Management Science", 1(1), 197-206.
- Defense, D. o. (May 2009). *DOD Architecture Framework Version 2.0*.
- DeLaurentis, D. A., & Callaway, R. K. (Nov. 2004). A system-of-systems perspective for public policy decisions. *Review of Policy Research*, Vol. 21(No. 6), pp. 829-837.
- DeLaurentis, D., Crossley, W., & Mane, M. (2011). Taxonomy to Guide SoS Decision-Making in Air Transportation Problems. *Journal of Aircraft*, Vol. 48(No. 3).
- DeStefano, G. (2004). *Agent Based Simulation SEAS Evaluation of DoDAF Architecture*. Air Force Institute of Technology.
- Dodaro, G. (2010). *Defense Acquisitions: Assessment of Selected Weapons Programs*. Washington, D.C.: overnment Accountability Office, U.S. Government.
- Domercant, J., & Mavris, D. (2010). Measuring the Architectural Complexity of Military System-of-Systems. *IEEE AC paper #1649 version2*.
- Dyer, D., & Chiou, P. (1984). An information-theoretic approach to incorporating prior information in binomial sampling. *Communications in Statistics - Theory and Methods*, vol. 13, pp. 2051 - 2083.
- E. HollnagelW. Woods, and N. LevesonD. (Hampshire). "Resilience Engineering: Concepts and Precepts." England: Ashgate Publishing Ltd.
- European Organization for the Safety of Air Navigation. (2007). *Architecture Evolution Plan*.



- Fiksel, J. (2006). Sustainability and resilience: toward a systems approach. *Sustainability Science, Practice, & Policy*, 2(2), 1-8.
- Folke, C., Carpenter, S., Walker, B., Scheffer, M., Elmqvist, T., Gunderson, L., et al. (2004). Regime Shifts, Resilience, and Biodiversity in Ecosystem Management. *Annual Review of Ecology, Evolution, and Systematics*, 35, 557-581.
- Friedman, N., & Koller, D. (July 2009). *Probabilistic Graphical Models: Principles and Techniques*. Cambridge: The MIT Press.
- Fry, D., & DeLaurentis, D. (June 2011). Measuring Net-Centricity. *Proceedings of the 2011 6th International Conference on System of Systems Engineering*. New Mexico, USA.
- Garlan, D., & Shaw, M. (1994). *An introduction to Software Architecture*. CMU software engineering institute technical report, SEI-94-TR-21.
- GarrettAnderson, S., Baron, N., Moreland, J., R.,. (2011). Managing the Interstitials, a System of Systems Framework Suited for the Ballistic Missile Defense System. "Systems Engineering", 14(1).
- Garvey, P. ., & Pinto, A. (2009). *Introduction to Functional Dependency Network Analysis*. Lexington, MA: MIT.
- Garvey, P., & Pinto, A. (2012). *Advanced Risk Analysis in Engineering Enterprise Systems*. CRC Press.
- Goodstein, L., Andersen, H., & Olsen, S. (1988). *Tasks, Errors and Mental Models*. London: Taylor & Francis.
- Gove, R., Sauser, B., & Ramirez-Marquez, J. (2007). Integration Maturity Metrics: Development of an Integration Readiness Level. *SSE\_S&EM\_004\_2007*, In Stevens Institute of Technology, School of Systems and Enterprises. Hoboken, NJ.
- Gray, B. (2009). *Review of Acquisition for the Secretary of State for Defence*. London, EN.: Ministry of Defence, British Commonwealth.
- Griending, K., & Mavris, D. (2011). Development of a DoDAF-based Executable Architecting Approach to Analyze System-of-Systems Alternatives. *IEEE AC paper #1389 version1*.
- Gualtieri, J., Roth, E., & Eggleston, R. (April 30 - May 4, 2000). Utilizing the abstraction hierarchy for role allocation and team structure design. *the 5th International Conference on Human Interaction with Complex Systems*. Urbana, IL.
- Hajdukiewicz, J., Vicente, K. J., Doyle, D. J., Milgram, P., & Burns, C. M. (2001). Modelling a medical environment: and ontology for integrated medical informatics design. *International Journal of Medical Informatics*, 62(1), 79-99.
- Han, S. ., & DeLaurentis, D. (2012). Acquisition Management for System-of-Systems: Requirement Evolution and Acquisition Strategy Planning. *9th Naval Postgraduate School Symposium*. Monterey, California.
- Han, S. Y., & DeLaurentis, D. (Sep. 2011). *Air Traffic Demand Forecast at a Commercial Airport using Bayesian Networks*. AIAA Paper 2011-6905.
- Han, S., Marais, K., & DeLaurentis, D. (2012). Evaluating System of Systems Resilience using Interdependency Analysis. *IEEE International Conference on Systems, Man, and Cybernetics*. Seoul, Korea.
- Hsu, J. (2009). Emergent Behavior of Systems-of-Systems. *INCOSE Mini-Conference*.
- Jacobson, K. (Sept.2010). *The Littoral Combat Ship (LCS) Surface Warfare (SUW) Module: Determining the Best Mix of Surface-to-Surface and Air-to-Surface Missiles*.
- Jain, P. (June 2011). Architecture Evolution and Evaluation (ArchEE) Capability. *Proceedings of the 2011 6th International Conference on System of System Engineering*. New Mexico, USA.
- Jensen, K., Kristensen, L., & Wells, L. (2007). *Coloured Petri Nets and CPN Tools for Modeling and Validation of Concurrent Systems*. Software Tools for Technology Transfer. (July 2007). *NextGen Air Transportation System Enterprise Architecture*.



- Kasse Initiatives. (2004). System Architectures. *NDIA CMMI Conference*.
- Kinnunen, M. (Feb.2006). *Complexity Measure for System Architecture Models*. MIT.
- KotegawaDeLaurentis, D.A., Sengstacken, A., Han, En-pei,T.,. (2008). Utilization of Network Theory for the Enhancement of ATO Air Route Forecast. "8th Aviation, Technology, Integration and Operations (ATIO)." Anchorage, Alaska: AIAA.
- Kotov, V. (1997). *Systems-of-Systems as Communicating Structures*. Hewlett Packard Computer Systems Laboratory Paper HPL-97-124, pp. 1–15.
- Laughlin, R. (2005). *A Different Universe: Reinventing Physics from the Bottom Down*. Basic Books.
- Liu, J., Li, J., & Kim, B. (2011). Bayesian reliability modeling of multi-level system with interdependent subsystems and components. *IEEE International Conference on Intelligence and Security Informatics*, (pp. pp. 252-257). Beijing, China.
- Madni, A., & Jackson, S. (2009). Towards a Conceptual Framework for Resilience Engineering. *IEEE Systems Journal*, Vol 3(No. 2), pp. 181-191.
- Maier, M. W. (1998). Architecting Principles for System-of-Systems. *Systems Engineering*, Vol. 1(No. 4), pp. 267-284.
- Mane, M., DeLaurentis, D., & Frazho, A. (2011). A Markov Perspective on Development Interdependencies in Networks of Systems. *Journal of Mechanical Design*, Vol. 133(No. 10).
- ManeCrossley, W.A.,M.,. (2012). Allocation and Design of Aircraft for On-Demand Air Transportation Operations. "Journal of Aircraft".
- Mangino, K., Bolczak, K., & Simons, M. (March 2008). *SWIM Evolution Strategy*. MITRE Corporation.
- Mankins, J. (1995). *Technology Readiness Levels: A white paper*. Advance Concepts Office, Office of Space Access and Technology: NASA.
- MarkowitzHarry. (1952). Portfolio selection. "Journal of Finance", 77-91.
- Masten, A. S. (2009). Ordinary Magic: Lessons from research on resilience in human development. *Education Canada*, 49(3), 28-32.
- Mohammad, A. J., Hutchison, D., & Sterbenz, J. P. (Nov. 2006). Poster: Towards Quantifying Metrics for Resilient and Survivable Networks. *14th IEEE International Conference on Network Protocols (ICNP 2006)*. Santa Barbara, California, USA.
- Neches, R., & Madni, A. (2012). Towards Affordably Adaptable and Effective Systems. *Journal of Systems Engineering*.
- Nishijima, K., Maes, M., Goyet, J., & Faber, M. H. (March 26-27, 2007). *Optimal Reliability of Components of Complex Systems using Hierarchical System Models*. special workshop on risk acceptance and risk communication, Stanford University.
- O'Rourke, R. (April 6, 2012). *Navy Littoral Combat Ship (LCS) Program: Background, Issues, and Options for Congress*. Congressional Research Service Report.
- OUSD(AT&L). (August 2008). *Systems Engineering Guide for System of Systems*. Washington, D.C.: Pentagon.
- Pflanz, M., & Levis, A. (March 2012). An Approach to Evaluating Resilience in Command and Control Architectures. *Systems Engineering and Architecting Conference on Systems Engineering Research (CSER)*. st. Louis, MO.
- Rausand, M., & Hoyland, A. (2004). *System Reliability Theory: Models, Statistical Methods, and Applications*. Wiley – Interscience.
- Reese, C., Johnson, V., Hama, M., & Wilson, A. (2005). *A Hierarchical Model for the Reliability of an Anti-aircraft Missile System*. UT MD Anderson Cancer Center Department of Biostatistics Working Paper Series.

- Sage, A., & Cuppan, C. (2001). On the Systems Engineering and Management of Systems of Systems and Federations of Systems. *Information, Knowledge, Systems Management*, Vol. 2(No. 4), pp. 325-345.
- Sheard, S., & Mostashari, A. (2008). A Framework for System Resilience Discussions. *Annual International Symposium of INCOSE*. Utrecht, Netherlands.
- Shinozuka, M., & Chang, S. (2004). *Evaluating the Disaster Resilience of Power Networks and Grids*. (Y. Okuyama, & S. Chang, Eds.) Heidelberg: Springer.
- Springer, M., & Thompson, W. (1966). The Distribution of Products of Independent Random Variables. *SIAM Journal on Applied Mathematics*, vol. 14, pp. 511-526.
- Stuart, D., Mattikalli, R., DeLaurentis, D., & Shah, J. (Sept.2011). *Boeing META II: Complexity and Adaptability*.
- Tan, W., Ramirez-Marquez, J., & Sauser, B. (2011). A probabilistic approach to system maturity assessment. *Systems Engineering*, Vol 14 No.3.
- Thompson, W., & Haynes, R. (1980). On the reliability, availability and bayes confidence intervals for multi component systems. *Naval Research Logistics Quarterly*, vol. 27, pp. 345-358.
- TutuncuKoenig, M., R., (2004). Robust Asset Allocation. "Annals of Operations Research", 157-187.
- Wagenhals, L., & Levis, A. (2008). Service Oriented Architectures, the DoD Architecture Framework 1.5 and Executable Architectures. *Systems Engineering*.
- Wang, R., & Dagli, C. (2011). Executable System Architecture using Systems Modeling Language in Conjunction with Colored Petri Nets in a Model Driven Systems Development Process. *System Engineering*, Vol.14(No.4), pp:383-408.
- Xu, N., Donohue, G., Laskey, K. B., & Chen, C. (June 2005). *Estimation of Delay Propagation in Aviation System using Bayesian Network*. Baltimore, MD: 6th USAEUROPE ATM Seminar.
- Yang, K., Chen, Y., Lu, Y., & Zhao, Q. (2010). The Study of Guided Emergent Behavior in System of Systems Requirement Analysis. *5th International Conference on System of Systems Engineering*.